Проект DRC

Разработчик Dinrus Group,

Виталий Кулич

http://github.com/DinrusGroup/DRC

Общая Цель Проекта:

Создать многоплатформенный компилятор для языка прогаммирования Динрус

Текущая Задача:

проанализировать существующий код и определить уже разработанные элементы для нового компилятора Динрус - ДРК (DRC).

модуль drc.Enums

```
импортирует common;

/// Перечень классов хранения.
перечень КлассХранения
{
  Нет          = 0,
  Абстрактный      = 1,
  Авто          = 1<<2,
  Конст         = 1<<3,
  Устаревший    = 1<<4,
  Экстерн        = 1<<5,
  Окончательный          = 1<<6,
  Инвариант     = 1<<7,
  Перепись      = 1<<8,
  Масштаб         = 1<<9,
  Статический        = 1<<10,
  Синхронизованный = 1<<11,
  Вхо            = 1<<12,
  Вых           = 1<<13,
  Реф           = 1<<14,
  Отложенный          = 1<<15,
  Вариадический      = 1<<16,
  Манифест      = 1<<17
}

/// Перечень атрибутов защиты.
перечень Защита
{
  Нет,
  Приватный/+    = 1+/,
  Защищённый/+ = 1<<1+/,
  Пакет/+    = 1<<2+/,
  Публичный/+     = 1<<3+/,
  Экспорт/+     = 1<<4+/
}

/// Перечень типов компоновки.
перечень ТипКомпоновки
{
  Нет,
  C,
```

```
  Cpp,
  D,
  Windows,
  Pascal,
  Система
}

/// Возвращает ткст для защ.
ткст вТкст(Защита защ)
{
  щит (защ)
  { вместо Защита З;
  равно З.Нет:       выдай "";
  равно З.Приватный:    выдай "private";
  равно З.Защищённый: выдай "protected";
  равно З.Пакет:    выдай "package";
  равно З.Публичный:     выдай "public";
  равно З.Экспорт:     выдай "export";
  дефолт:
    подтверди(0);
  }
}

/// Возвращает ткст для защ.
ткст вРусТкст(Защита защ)
{
  щит (защ)
  { вместо Защита З;
  равно З.Нет:       выдай "";
  равно З.Приватный:    выдай "прив";
  равно З.Защищённый: выдай "защ";
  равно З.Пакет:    выдай "пак";
  равно З.Публичный:     выдай "пуб";
  равно З.Экспорт:     выдай "эксп";
  дефолт:
    подтверди(0);
  }
}

/// Возвращает ткст класса хранения. Может быть установлен только 1 бит.
ткст вТкст(КлассХранения кхр)
{
  щит (кхр)
  { вместо КлассХранения КХ;
  равно КХ.Абстрактный:      выдай "abstract";
  равно КХ.Авто:          выдай "auto";
  равно КХ.Конст:         выдай "const";
  равно КХ.Устаревший:    выдай "deprecated";
  равно КХ.Экстерн:        выдай "extern";
  равно КХ.Окончательный:         выдай "final";
  равно КХ.Инвариант:     выдай "invariant";
  равно КХ.Перепись:      выдай "override";
  равно КХ.Масштаб:         выдай "scope";
  равно КХ.Статический:        выдай "static";
  равно КХ.Синхронизованный: выдай "synchronized";
  равно КХ.Вхо:            выдай "in";
  равно КХ.Вых:           выдай "out";
  равно КХ.Реф:           выдай "ref";
  равно КХ.Отложенный:          выдай "lazy";
  равно КХ.Вариадический:      выдай "variadic";
  равно КХ.Манифест:      выдай "manifest";
  дефолт:
    подтверди(0);
```

```
  }
}

ткст вРусТкст(КлассХранения кхр)
{
  щит (кхр)
  { вместо КлассХранения КХ;
  равно КХ.Абстрактный:     выдай "абстр";
  равно КХ.Авто:          выдай "авто";
  равно КХ.Конст:         выдай "конст";
  равно КХ.Устаревший:    выдай "устар";
  равно КХ.Экстерн:        выдай "внеш";
  равно КХ.Окончательный:        выдай "фин";
  равно КХ.Инвариант:     выдай "неизм";
  равно КХ.Перепись:      выдай "перен";
  равно КХ.Масштаб:         выдай "масшт";
  равно КХ.Статический:       выдай "стат";
  равно КХ.Синхронизованный: выдай "синх";
  равно КХ.Вхо:            выдай "вхо";
  равно КХ.Вых:           выдай "вых";
  равно КХ.Реф:           выдай "ссыл";
  равно КХ.Отложенный:         выдай "отлож";
  равно КХ.Вариадический:     выдай "вариад";
  равно КХ.Манифест:      выдай "манифест";
  дефолт:
    подтверди(0);
  }
}

/// Возвращает тксты for кхр. Any число of bits may be установи.
ткст[] вТксты(КлассХранения кхр)
{
  ткст[] результат;
  при (авто i = КлассХранения.max; i; i >>= 1)
    если (кхр & i)
      результат ~= вТкст(i);
  выдай результат;
}

/// Возвращает ткст for ltype.
ткст вТкст(ТипКомпоновки ltype)
{
  щит (ltype)
  { вместо ТипКомпоновки ТК;
  равно ТК.Нет:    выдай "";
  равно ТК.С:       выдай "C";
  равно ТК.Cpp:     выдай "Cpp";
  равно ТК.D:       выдай "D";
  равно ТК.Windows: выдай "Windows";
  равно ТК.Pascal:  выдай "Pascal";
  равно ТК.Система:  выдай "System";
  дефолт:
    подтверди(0);
  }
}

ткст вРусТкст(ТипКомпоновки ltype)
{
  щит (ltype)
  { вместо ТипКомпоновки ТК;
  равно ТК.Нет:    выдай "";
  равно ТК.С:       выдай "Си";
  равно ТК.Cpp:     выдай "Сипп";
```

```
  равно ТК.D:       выдай "Ди";
  равно ТК.Windows: выдай "Вин";
  равно ТК.Pascal:  выдай "Паскаль";
  равно ТК.Система:  выдай "Система";
  дефолт:
    подтверди(0);
  }
}
```

Пакет AST (Абстрактное Семантическое Дерево)

модуль drc.ast.Declaration;

```
 импортирует drc.ast.Node,  drc.Enums;

 /// Корневой класс всех деклараций.
 абстр класс Декларация : Узел

бул естьТело;

 // Члены, соответствующие семантической фазе.
КлассХранения кхр; /// Классы сохранения данной декларации.
Защита защ; /// Атрибут защиты данной декларации.

фин бул статический_ли();

фин бул публичный_ли();

фин проц установиКлассХранения(КлассХранения кхр);

фин проц установиЗащиту(Защита защ);

переп абстр Декларация копируй();
```

модуль drc.ast.Declarations;

```
пуб импортирует drc.ast.Declaration;
импортирует drc.ast.Node,
       drc.ast.Expression,
       drc.ast.Types,
       drc.ast.Statements,
       drc.ast.Parameters,
       drc.ast.NodeCopier,
drc.lexer.IdTable,
drc.semantic.Symbols,
drc.Enums,common;

класс СложнаяДекларация : Декларация
{
  сам();

  проц  opCatAssign(Декларация d);

  проц  opCatAssign(СложнаяДекларация ds);

  Декларация[] деклы();

  проц  деклы(Декларация[] деклы);
```

```
  внедри(методКопирования);
}

/// Единичная точка с запятой.
класс ПустаяДекларация : Декларация
{
  сам();

  внедри(методКопирования);
}

/// Нелегальным декларациям соответствуют все семы,
/// которые не начинают ДефинициюДекларации.
/// See_Also: drc.lexer.Token.семаНачалаДеклДеф_ли()
класс НелегальнаяДекларация : Декларация
{
  сам();
  внедри(методКопирования);
}

/// ПКИ = полностью "квалифицированное" имя
вместо Идентификатор*[] ПКИМодуля; // Идентификатор(.Идентификатор)*

класс ДекларацияМодуля : Декларация
{
  Идентификатор* имяМодуля;
  Идентификатор*[] пакеты;
  сам(ПКИМодуля пкиМодуля);

  ткст дайПКН();

  ткст дайИмя();

  ткст дайИмяПакета(сим разделитель);

  внедри(методКопирования);
}

класс ДекларацияИмпорта : Декларация
{
  прив вместо Идентификатор*[] Иды;
  ПКИМодуля[] пкиМодулей;
  Иды алиасыМодуля;
  Иды связанныеИмена;
  Иды связанныеАлиасы;

  сам(ПКИМодуля[] пкиМодулей, Иды алиасыМодуля, Иды связанныеИмена, Иды
связанныеАлиасы, бул статический_ли);

  сим[][] дайПКНМодуля(сим разделитель);

  внедри(методКопирования);
}

класс ДекларацияАлиаса : Декларация
{
  Декларация декл;
  сам(Декларация декл)
  {
    внедри(установить_вид);
    добавьОтпрыск(декл);
    сам.декл = декл;
```

```
  }

  внедри(методКопирования);
}

класс ДекларацияТипдефа : Декларация
{
  Декларация декл;
  сам(Декларация декл);
  внедри(методКопирования);
}

класс ДекларацияПеречня : Декларация
{
  Идентификатор* имя;
  УзелТипа типОснова;
  ДекларацияЧленаПеречня[] члены;
  сам(Идентификатор* имя, УзелТипа типОснова, ДекларацияЧленаПеречня[] члены,
бул естьТело);

  Перечень символ;

  внедри(методКопирования);
}

класс ДекларацияЧленаПеречня : Декларация
{
  УзелТипа тип; // D 2.0
  Идентификатор* имя;
  Выражение значение;
  сам(Идентификатор* имя, Выражение значение);

  // D 2.0
  сам(УзелТипа тип, Идентификатор* имя, Выражение значение);

  ЧленПеречня символ;

  внедри(методКопирования);
}

класс ДекларацияШаблона : Декларация
{
  Идентификатор* имя;
  ПараметрыШаблона шпарамы;
  Выражение констрейнт; // D 2.0
  СложнаяДекларация деклы;
  сам(Идентификатор* имя, ПараметрыШаблона шпарамы, Выражение констрейнт,
СложнаяДекларация деклы);

  Шаблон символ; /// Шаблонный символ для данной декларации.

  внедри(методКопирования);
}

// Примечание: шпарамы закомментированы, поскольку Парсер
//       оборачивает декларации шаблонными параметрами внутри
ДеклорацииШаблона.
//

абстр класс ДекларацияАгрегата : Декларация
{
  Идентификатор* имя;
//   ПараметрыШаблона шпарамы;
```

```
  СложнаяДекларация деклы;
  сам(Идентификатор* имя, /+ПараметрыШаблона шпарамы, +/СложнаяДекларация
деклы);
}

класс ДекларацияКласса : ДекларацияАгрегата
{
  ТипКлассОснова[] основы;
  сам(Идентификатор* имя, /+ПараметрыШаблона шпарамы, +/ТипКлассОснова[]
основы, СложнаяДекларация деклы);

  Класс символ; /// Символ класса данной декларации.

  внедри(методКопирования);
}

класс ДекларацияИнтерфейса : ДекларацияАгрегата
{
  ТипКлассОснова[] основы;
  сам(Идентификатор* имя, /+ПараметрыШаблона шпарамы, +/ТипКлассОснова[]
основы, СложнаяДекларация деклы)
  {
    предок(имя, /+шпарамы, +/деклы);
    внедри(установить_вид);
//      добавьОтпрыск(шпарамы);
    добавьОпцОтпрыски(основы);
    добавьОпцОтпрыск(деклы);

    сам.основы = основы;
  }

  вместо drc.semantic.Symbols.Интерфейс Интерфейс;

  Интерфейс символ; /// Символ интерфейса данной декларации.

  внедри(методКопирования);
}

класс ДекларацияСтруктуры : ДекларацияАгрегата
{
  бцел размерРаскладки;
  сам(Идентификатор* имя, /+ПараметрыШаблона шпарамы, +/СложнаяДекларация
деклы);

  проц  установиРазмерРаскладки(бцел размерРаскладки);

  Структура символ; /// Символ структуры данной декларации.

  внедри(методКопирования);
}

класс ДекларацияСоюза : ДекларацияАгрегата
{
  сам(Идентификатор* имя, /+ПараметрыШаблона шпарамы, +/СложнаяДекларация
деклы);

  Союз символ; /// Символ союза данной декларации.

  внедри(методКопирования);
}

класс ДекларацияКонструктора : Декларация
{
```

```
  Параметры парамы;
  ИнструкцияТелаФункции телоФунк;
  сам(Параметры парамы, ИнструкцияТелаФункции телоФунк);
  внедри(методКопирования);
}

класс ДекларацияСтатическогоКонструктора : Декларация
{
  ИнструкцияТелаФункции телоФунк;
  сам(ИнструкцияТелаФункции телоФунк);
  внедри(методКопирования);
}

класс ДекларацияДеструктора : Декларация
{
  ИнструкцияТелаФункции телоФунк;
  сам(ИнструкцияТелаФункции телоФунк);
  внедри(методКопирования);
}

класс ДекларацияСтатическогоДеструктора : Декларация
{
  ИнструкцияТелаФункции телоФунк;
  сам(ИнструкцияТелаФункции телоФунк);
  внедри(методКопирования);
}

класс ДекларацияФункции : Декларация
{
  УзелТипа типВозврата;
  Идентификатор* имя;
//   ПараметрыШаблона шпарамы;
  Параметры парамы;
  ИнструкцияТелаФункции телоФунк;
  ТипКомпоновки типКомпоновки;
  бул нельзяИнтерпретировать = нет;
  сам(УзелТипа типВозврата, Идентификатор* имя,/+ ПараметрыШаблона шпарамы,+/
       Параметры парамы, ИнструкцияТелаФункции телоФунк);

  проц  установиТипКомпоновки(ТипКомпоновки типКомпоновки);

  бул вВидеШаблона_ли();

  внедри(методКопирования);
}

/// ДекларацияПеременных := Тип? Идентификатор ("=" Init)? ("," Идентификатор
("=" Init)?)* ";"
класс ДекларацияПеременных : Декларация
{
  УзелТипа узелТипа;
  Идентификатор*[] имена;
  Выражение[] иниты;
  сам(УзелТипа узелТипа, Идентификатор*[] имена, Выражение[] иниты);

  ТипКомпоновки типКомпоновки;

  проц  установиТипКомпоновки(ТипКомпоновки типКомпоновки);

  Переменная[] переменные;

  внедри(методКопирования);
}
```

```
//Invariant
класс ДекларацияИнварианта : Декларация
{
  ИнструкцияТелаФункции телоФунк;
  сам(ИнструкцияТелаФункции телоФунк);
  внедри(методКопирования);
}

//Unittest
класс ДекларацияЮниттеста : Декларация
{
  ИнструкцияТелаФункции телоФунк;
  сам(ИнструкцияТелаФункции телоФунк);
  внедри(методКопирования);
}

абстр класс ДекларацияУсловнойКомпиляции : Декларация
{
  Сема* спец;
  Сема* услов;
  Декларация деклы, деклыИначе;

  сам(Сема* спец, Сема* услов, Декларация деклы, Декларация деклыИначе);

  бул определение_ли();

  бул условие_ли();

  /// Ветвь, в которой компилируется.
  Декларация компилированныеДеклы;
}

//Debug
класс ДекларацияОтладки : ДекларацияУсловнойКомпиляции
{
  сам(Сема* спец, Сема* услов, Декларация деклы, Декларация деклыИначе);
  внедри(методКопирования);
}

//Version
класс ДекларацияВерсии : ДекларацияУсловнойКомпиляции
{
  сам(Сема* спец, Сема* услов, Декларация деклы, Декларация деклыИначе);
  внедри(методКопирования);
}

//Static Если
класс ДекларацияСтатическогоЕсли : Декларация
{
  Выражение условие;
  Декларация деклыЕсли, деклыИначе;
  сам(Выражение условие, Декларация деклыЕсли, Декларация деклыИначе);
  внедри(методКопирования);
}

//Static Подтверди
класс ДекларацияСтатическогоПодтверди : Декларация
{
  Выражение условие, сообщение;
  сам(Выражение условие, Выражение сообщение);
  внедри(методКопирования);
}
```

```
//New
класс ДекларацияНов : Декларация
{
  Параметры парамы;
  ИнструкцияТелаФункции телоФунк;
  сам(Параметры парамы, ИнструкцияТелаФункции телоФунк);
  внедри(методКопирования);
}

//Delete
класс ДекларацияУдали : Декларация
{
  Параметры парамы;
  ИнструкцияТелаФункции телоФунк;
  сам(Параметры парамы, ИнструкцияТелаФункции телоФунк);
  внедри(методКопирования);
}

абстр класс ДекларацияАтрибута : Декларация
{
  Декларация деклы;
  сам(Декларация деклы);
}

класс ДекларацияЗащиты : ДекларацияАтрибута
{
  Защита защ;
  сам(Защита защ, Декларация деклы);
  внедри(методКопирования);
}

класс ДекларацияКлассаХранения : ДекларацияАтрибута
{
  КлассХранения классХранения;
  сам(КлассХранения классХранения, Декларация декл);
  внедри(методКопирования);
}

класс ДекларацияКомпоновки : ДекларацияАтрибута
{
  ТипКомпоновки типКомпоновки;
  сам(ТипКомпоновки типКомпоновки, Декларация деклы);
  внедри(методКопирования);
}

класс ДекларацияРазложи : ДекларацияАтрибута
{
  цел размер;
  сам(цел размер, Декларация деклы);
  внедри(методКопирования);
}

класс ДекларацияПрагмы : ДекларацияАтрибута
{
  Идентификатор* идент;
  Выражение[] арги;
  сам(Идентификатор* идент, Выражение[] арги, Декларация деклы);
  внедри(методКопирования);
}

класс ДекларацияСмеси : Декларация
{
```

```
  /// IdExpression := ВыражениеИдентификатор | ВыражениеЭкземплярШаблона
  /// ВнедриTemplate := IdExpression ("." IdExpression)*
  Выражение выражШаблон;
  Идентификатор* идентСмеси; /// Optional внедри identeслиier.
  Выражение аргумент; /// "внедри" "(" ВыражениеПрисвой ")"
  Декларация деклы; /// Инициализируется на семантической фазе.

  сам(Выражение выражШаблон, Идентификатор* идентСмеси);

  сам(Выражение аргумент);

  бул выражениеСмеси_ли();

  внедри(методКопирования);
}
```

модуль drc.ast.DefaultVisitor;

```
/// Генерирует рабочий код для посещения предоставленных членов.
private ткст создайКод(ВидУзла видУзла);

/// Генерирует методы визита по умолчанию.
///
/// Напр.:
/// ---
/// override типВозврата!("ДекларацияКласса") посети(ДекларацияКласса n)
/// { /* Код, посещения подузлов... */ return n; }
/// ---
ткст генерируйДефМетодыВизита();
// pragma(сооб, генерируйДефМетодыВизита());

/// Этот класс предоставляет методы по умолчанию для обхода
/// узлов и их подузлов.
class ДефолтныйВизитёр : Визитёр
{
  // Закоментируйте, если появится много ошибок.
  mixin(генерируйДефМетодыВизита());
}
```

модуль drc.ast.Expression;

```
import drc.ast.Node;
import drc.semantic.Types,
       drc.semantic.Symbol;
import common;

/// Корневой классс всех выражений.
abstract class Выражение : Узел
{
  Тип тип; /// Семантический тип данного выражения.
  Символ символ;

  this()
  {
    super(КатегорияУзла.Выражение);
  }

  /// Возвращает да, если член 'тип' не равен null.
  бул естьТип()
  {
```

```
    return тип !is null;
  }

  /// Возвращает да, если член 'символ' не равен null.
  бул естьСимвол()
  {
    return символ !is null;
  }

  override abstract Выражение копируй();
}
```

module drc.ast.Expressions;

```
public import drc.ast.Expression;
import drc.ast.Node,
       drc.ast.Types,
       drc.ast.Declarations,
       drc.ast.Statements,
       drc.ast.Parameters,
       drc.ast.NodeCopier;
import drc.lexer.Identifier;
import drc.semantic.Types;
import common;

class НелегальноеВыражение : Выражение
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

abstract class БинарноеВыражение : Выражение
{
  Выражение лв; /// Левосторонняя сторона выражения.
  Выражение пв; /// Правосторонняя сторона выражения.
  Сема* лекс;

  this(Выражение лв, Выражение пв, Сема* лекс)
  {
    добавьОтпрыски([лв, пв]);
    this.лв = лв;
    this.пв = пв;
    this.лекс = лекс;
  }
  mixin(бинарноеВыражениеМетодаКопирования);
}

class ВыражениеУсловия : БинарноеВыражение
{
  Выражение условие;
  this(Выражение условие, Выражение левый, Выражение правый, Сема* лекс)
  {
    добавьОтпрыск(условие);
    super(левый, правый, лекс);
    mixin(установить_вид);
    this.условие = условие;
  }
  mixin(методКопирования);
}
```

```
class ВыражениеЗапятая : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеИлиИли : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеИИ : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеИли : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеИИли : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеИ : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

abstract class ВыражениеСравни : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
  }
}
```

```
class ВыражениеРавно : ВыражениеСравни
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

/// Выражение "!"? "is" Выражение
class ВыражениеРавенство : ВыражениеСравни
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеОтнош : ВыражениеСравни
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеВхо : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеЛСдвиг : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеПСдвиг : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеБПСдвиг : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}
```

```
class ВыражениеПлюс : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеМинус : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеСоедини : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеУмножь : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеДели : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеМод : БинарноеВыражение
{
  this(Выражение левый, Выражение правый, Сема* лекс)
  {
    super(левый, правый, лекс);
    mixin(установить_вид);
  }
}

class ВыражениеПрисвой : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
```

```
class ВыражениеПрисвойЛСдвиг : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
class ВыражениеПрисвойПСдвиг : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
class ВыражениеПрисвойБПСдвиг : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
class ВыражениеПрисвойИли : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
class ВыражениеПрисвойИ : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
class ВыражениеПрисвойПлюс : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
class ВыражениеПрисвойМинус : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
class ВыражениеПрисвойДел : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
```

```
}
class ВыражениеПрисвойУмн : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
class ВыражениеПрисвойМод : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
class ВыражениеПрисвойИИли : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}
class ВыражениеПрисвойСоед : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}

/// ВыражениеТочка := Выражение '.' Выражение
class ВыражениеТочка : БинарноеВыражение
{
  this(Выражение левый, Выражение правый)
  {
    super(левый, правый, null);
    mixin(установить_вид);
  }
}

/*++++++++++++++++++++
+ Unary Expressions: +
++++++++++++++++++++*/

abstract class УнарноеВыражение : Выражение
{
  Выражение в; // TODO: rename 'e' в 'следщ', 'unary', 'выр' or 'sube' etc.
  this(Выражение в)
  {
    добавьОтпрыск(в);
    this.в = в;
  }
  mixin(унарноеВыражениеМетодаКопирования);
}

class ВыражениеАдрес : УнарноеВыражение
{
  this(Выражение в)
  {
```

```
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеПреИнкр : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеПреДекр : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеПостИнкр : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеПостДекр : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеДереф : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеЗнак : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }

  бул положит_ли()
  {
    assert(начало !is null);
    return начало.вид == ТОК.Плюс;
  }
```

```
  бул отриц_ли()
  {
    assert(начало !is null);
    return начало.вид == ТОК.Минус;
  }
}

class ВыражениеНе : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеКомп : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеВызов : УнарноеВыражение
{
  Выражение[] арги;
  this(Выражение в, Выражение[] арги)
  {
    super(в);
    mixin(установить_вид);
    добавьОпцОтпрыски(арги);
    this.арги = арги;
  }
}

class ВыражениеНов : /*Unary*/Выражение
{
  Выражение[] новАрги;
  УзелТипа тип;
  Выражение[] кторАрги;
  this(/*Выражение в, */Выражение[] новАрги, УзелТипа тип, Выражение[]
кторАрги)
  {
    /*super(в);*/
    mixin(установить_вид);
    добавьОпцОтпрыски(новАрги);
    добавьОтпрыск(тип);
    добавьОпцОтпрыски(кторАрги);
    this.новАрги = новАрги;
    this.тип = тип;
    this.кторАрги = кторАрги;
  }
  mixin(методКопирования);
}

class ВыражениеНовАнонКласс : /*Unary*/Выражение
{
  Выражение[] новАрги;
  ТипКлассОснова[] основы;
  Выражение[] кторАрги;
```

```
  СложнаяДекларация деклы;
  this(/*Выражение в, */Выражение[] новАрги, ТипКлассОснова[] основы,
Выражение[] кторАрги, СложнаяДекларация деклы)
  {
    /*super(в);*/
    mixin(установить_вид);
    добавьОпцОтпрыски(новАрги);
    добавьОпцОтпрыски(основы);
    добавьОпцОтпрыски(кторАрги);
    добавьОтпрыск(деклы);

    this.новАрги = новАрги;
    this.основы = основы;
    this.кторАрги = кторАрги;
    this.деклы = деклы;
  }
  mixin(методКопирования);
}

class ВыражениеУдали : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеКаст : УнарноеВыражение
{
  УзелТипа тип;
  this(Выражение в, УзелТипа тип)
  {
    добавьОтпрыск(тип); // Add тип before super().
    super(в);
    mixin(установить_вид);
    this.тип = тип;
  }
  mixin(методКопирования);
}

class ВыражениеИндекс : УнарноеВыражение
{
  Выражение[] арги;
  this(Выражение в, Выражение[] арги)
  {
    super(в);
    mixin(установить_вид);
    добавьОтпрыски(арги);
    this.арги = арги;
  }
  mixin(методКопирования);
}

class ВыражениеСрез : УнарноеВыражение
{
  Выражение левый, правый;
  this(Выражение в, Выражение левый, Выражение правый)
  {
    super(в);
    mixin(установить_вид);
    assert(левый ? (правый !is null) : правый is null);
    if (левый)
```

```
      добавьОтпрыски([левый, правый]);

    this.левый = левый;
    this.правый = правый;
  }
  mixin(методКопирования);
}

/// Модуль Масштаб operator: '.'
(ВыражениеИдентификатор|ВыражениеЭкземплярШаблона)
class ВыражениеМасштабМодуля : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    assert(в.вид == ВидУзла.ВыражениеИдентификатор ||
           в.вид == ВидУзла.ВыражениеЭкземплярШаблона
    );
    mixin(установить_вид);
  }
}

/*++++++++++++++++++++++
+ Primary Expressions: +
++++++++++++++++++++++*/

class ВыражениеИдентификатор : Выражение
{
  Идентификатор* идент;
  this(Идентификатор* идент)
  {
    mixin(установить_вид);
    this.идент = идент;
  }

  Сема* идСема()
  {
    assert(начало !is null);
    return начало;
  }

  mixin(методКопирования);
}

class ВыражениеЭкземплярШаблона : Выражение
{
  Идентификатор* идент;
  АргументыШаблона шарги;
  this(Идентификатор* идент, АргументыШаблона шарги)
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(шарги);
    this.идент = идент;
    this.шарги = шарги;
  }

  Сема* идСема()
  {
    assert(начало !is null);
    return начало;
  }

  mixin(методКопирования);
```

```
}

class ВыражениеСпецСема : Выражение
{
  Сема* особаяСема;
  this(Сема* особаяСема)
  {
    mixin(установить_вид);
    this.особаяСема = особаяСема;
  }

  Выражение значение; /// Выражение, созданное на семантической фазе.

  mixin(методКопирования);
}

class ВыражениеЭтот : Выражение
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

class ВыражениеСупер : Выражение
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

class ВыражениеНуль : Выражение
{
  this()
  {
    mixin(установить_вид);
  }

  this(Тип тип)
  {
    this();
    this.тип = тип;
  }

  mixin(методКопирования);
}

class ВыражениеДоллар : Выражение
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

class БулевоВыражение : Выражение
{
  ЦелВыражение значение; /// ЦелВыражение of тип бул.
```

```
  this(бул значение)
  {
    mixin(установить_вид);
    // Some semantic computation here.
    this.значение = new ЦелВыражение(значение, Типы.Бул);
    this.тип = Типы.Бул;
  }

  бул вБул()
  {
    assert(начало !is null);
    return начало.вид == ТОК.Истина ? да : нет;
  }

  mixin(методКопирования);
}

class ЦелВыражение : Выражение
{
  бдол число;

  this(бдол число, Тип тип)
  {
    mixin(установить_вид);
    this.число = число;
    this.тип = тип;
  }

  this(Сема* сема)
  {
    // Some semantic computation here.
    auto тип = Типы.Цел; // Should be most common case.
    switch (сема.вид)
    {
    // case ТОК.Цел32:
    //   тип = Типы.Цел; break;
    case ТОК.Бцел32:
      тип = Типы.Бцел; break;
    case ТОК.Цел64:
      тип = Типы.Дол; break;
    case ТОК.Бцел64:
      тип = Типы.Бдол; break;
    default:
      assert(сема.вид == ТОК.Цел32);
    }
    this(сема.бдол_, тип);
  }

  mixin(методКопирования);
}

class ВыражениеРеал : Выражение
{
  реал число;

  this(реал число, Тип тип)
  {
    mixin(установить_вид);
    this.число = число;
    this.тип = тип;
  }

  this(Сема* сема)
```

```
  {
    // Some semantic computation here.
    auto тип = Типы.Дво; // Most common case?
    switch (сема.вид)
    {
    case ТОК.Плав32:
      тип = Типы.Плав; break;
    // case ТОК.Плав64:
    //   тип = Типы.Дво; break;
    case ТОК.Плав80:
      тип = Типы.Реал; break;
    case ТОК.Мнимое32:
      тип = Типы.Вплав; break;
    case ТОК.Мнимое64:
      тип = Типы.Вдво; break;
    case ТОК.Мнимое80:
      тип = Типы.Вреал; break;
    default:
      assert(сема.вид == ТОК.Плав64);
    }
    this(сема.реал_, тип);
  }

  mixin(методКопирования);
}


/// Этот выражение holds a complex число.
/// It is only created in the semantic phase.
class ВыражениеКомплекс : Выражение
{
  креал число;
  this(креал число, Тип тип)
  {
    mixin(установить_вид);
    this.число = число;
    this.тип = тип;
  }
  mixin(методКопирования);
}

class ВыражениеСим : Выражение
{
  ЦелВыражение значение; // ЦелВыражение of тип Сим/Шим/Дим.
//  дим символ; // Replaced by значение.
  this(дим символ)
  {
    mixin(установить_вид);
//    this.символ = символ;
    // Some semantic computation here.
    if(символ <= 0xFF)
      this.тип = Типы.Сим;
    else if(символ <= 0xFFFF)
      this.тип = Типы.Шим;
    else
      this.тип = Типы.Дим;

    this.значение = new ЦелВыражение(символ, this.тип);
  }
  mixin(методКопирования);
}

class ТекстовоеВыражение : Выражение
```

```
{
  ббайт[] ткт;   /// The ткст данные.
  Тип типСим; /// The символ тип of the ткст.

  this(ббайт[] ткт, Тип типСим)
  {
    mixin(установить_вид);
    this.ткт = ткт;
    this.типСим = типСим;
    this.тип = new СМассивТип(типСим, ткт.length);
  }

  this(ткст ткт)
  {
    this(cast(ббайт[])ткт, Типы.Сим);
  }

  this(шим[] ткт)
  {
    this(cast(ббайт[])ткт, Типы.Шим);
  }

  this(дим[] ткт)
  {
    this(cast(ббайт[])ткт, Типы.Дим);
  }

  /// Возвращает ткст excluding the terminating 0.
  ткст дайТекст()
  {
    // TODO: convert в ткст if типСим !is Типы.Сим.
    return cast(сим[])ткт[0..$-1];
  }

  mixin(методКопирования);
}

class ВыражениеЛитералМассива : Выражение
{
  Выражение[] значения;
  this(Выражение[] значения)
  {
    mixin(установить_вид);
    добавьОпцОтпрыски(значения);
    this.значения = значения;
  }
  mixin(методКопирования);
}

class ВыражениеЛитералАМассива : Выражение
{
  Выражение[] ключи, значения;
  this(Выражение[] ключи, Выражение[] значения)
  {
    assert(ключи.length == значения.length);
    mixin(установить_вид);
    foreach (i, key; ключи)
      добавьОтпрыски([key, значения[i]]);
    this.ключи = ключи;
    this.значения = значения;
  }
  mixin(методКопирования);
}
```

```
class ВыражениеПодтверди : Выражение
{
  Выражение выр, сооб;
  this(Выражение выр, Выражение сооб)
  {
    mixin(установить_вид);
    добавьОтпрыск(выр);
    добавьОпцОтпрыск(сооб);
    this.выр = выр;
    this.сооб = сооб;
  }
  mixin(методКопирования);
}

class ВыражениеСмесь : Выражение
{
  Выражение выр;
  this(Выражение выр)
  {
    mixin(установить_вид);
    добавьОтпрыск(выр);
    this.выр = выр;
  }
  mixin(методКопирования);
}

class ВыражениеИмпорта : Выражение
{
  Выражение выр;
  this(Выражение выр)
  {
    mixin(установить_вид);
    добавьОтпрыск(выр);
    this.выр = выр;
  }
  mixin(методКопирования);
}

class ВыражениеТипа : Выражение
{
  УзелТипа тип;
  this(УзелТипа тип)
  {
    mixin(установить_вид);
    добавьОтпрыск(тип);
    this.тип = тип;
  }
  mixin(методКопирования);
}

class ВыражениеИдТипаТочка : Выражение
{
  УзелТипа тип;
  Идентификатор* идент;
  this(УзелТипа тип, Идентификатор* идент)
  {
    mixin(установить_вид);
    добавьОтпрыск(тип);
    this.тип = тип;
    this.идент = идент;
  }
  mixin(методКопирования);
```

```
}

class ВыражениеИдТипа : Выражение
{
  УзелТипа тип;
  this(УзелТипа тип)
  {
    mixin(установить_вид);
    добавьОтпрыск(тип);
    this.тип = тип;
  }
  mixin(методКопирования);
}

class ВыражениеЯвляется : Выражение
{
  УзелТипа тип;
  Идентификатор* идент;
  Сема* опцСема, спецСема;
  УзелТипа типСпец;
  ПараметрыШаблона шпарамы; // D 2.0
  this(УзелТипа тип, Идентификатор* идент, Сема* опцСема, Сема* спецСема,
       УзелТипа типСпец, typeof(шпарамы) шпарамы)
  {
    mixin(установить_вид);
    добавьОтпрыск(тип);
    добавьОпцОтпрыск(типСпец);
  version(D2)
    добавьОпцОтпрыск(шпарамы);
    this.тип = тип;
    this.идент = идент;
    this.опцСема = опцСема;
    this.спецСема = спецСема;
    this.типСпец = типСпец;
    this.шпарамы = шпарамы;
  }
  mixin(методКопирования);
}

class ВыражениеЛитералФункции : Выражение
{
  УзелТипа типВозврата;
  Параметры парамы;
  ИнструкцияТелаФункции телоФунк;

  this()
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(типВозврата);
    добавьОпцОтпрыск(парамы);
    добавьОтпрыск(телоФунк);
  }

  this(УзелТипа типВозврата, Параметры парамы, ИнструкцияТелаФункции
телоФунк)
  {
    this.типВозврата = типВозврата;
    this.парамы = парамы;
    this.телоФунк = телоФунк;
    this();
  }

  this(ИнструкцияТелаФункции телоФунк)
```

```
  {
    this.телоФунк = телоФунк;
    this();
  }

  mixin(методКопирования);
}

/// ParenthesisExpression := "(" Выражение ")"
class ВыражениеРодит : Выражение
{
  Выражение следщ;
  this(Выражение следщ)
  {
    mixin(установить_вид);
    добавьОтпрыск(следщ);
    this.следщ = следщ;
  }
  mixin(методКопирования);
}

// version(D2)
// {
class ВыражениеТрактовки : Выражение
{
  Идентификатор* идент;
  АргументыШаблона шарги;
  this(typeof(идент) идент, typeof(шарги) шарги)
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(шарги);
    this.идент = идент;
    this.шарги = шарги;
  }
  mixin(методКопирования);
}
// }

class ВыражениеИницПроц : Выражение
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

class ВыражениеИницМассива : Выражение
{
  Выражение[] ключи;
  Выражение[] значения;
  this(Выражение[] ключи, Выражение[] значения)
  {
    assert(ключи.length == значения.length);
    mixin(установить_вид);
    foreach (i, key; ключи)
    {
      добавьОпцОтпрыск(key); // The key is optional in ArrayInitializers.
      добавьОтпрыск(значения[i]);
    }
    this.ключи = ключи;
    this.значения = значения;
  }
```

```
  mixin(методКопирования);
}

class ВыражениеИницСтруктуры : Выражение
{
  Идентификатор*[] иденты;
  Выражение[] значения;
  this(Идентификатор*[] иденты, Выражение[] значения)
  {
    assert(иденты.length == значения.length);
    mixin(установить_вид);
    добавьОпцОтпрыски(значения);
    this.иденты = иденты;
    this.значения = значения;
  }
  mixin(методКопирования);
}

class ВыражениеТипАсм : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеСмещениеАсм : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеСегАсм : УнарноеВыражение
{
  this(Выражение в)
  {
    super(в);
    mixin(установить_вид);
  }
}

class ВыражениеАсмПослеСкобки : УнарноеВыражение
{
  Выражение e2; /// Выражение in brackets: в [ e2 ]
  this(Выражение в, Выражение e2)
  {
    super(в);
    mixin(установить_вид);
    добавьОтпрыск(e2);
    this.e2 = e2;
  }
  mixin(методКопирования);
}

class ВыражениеАсмСкобка : Выражение
{
  Выражение в;
  this(Выражение в)
  {
```

```
    mixin(установить_вид);
    добавьОтпрыск(в);
    this.в = в;
  }
  mixin(методКопирования);
}

class ВыражениеЛокальногоРазмераАсм : Выражение
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

class ВыражениеАсмРегистр : Выражение
{
  Идентификатор* регистр;
  цел число; // ST(0) - ST(7) or FS:0, FS:4, FS:8
  this(Идентификатор* регистр, цел число = -1)
  {
    mixin(установить_вид);
    this.регистр = регистр;
    this.число = число;
  }
  mixin(методКопирования);
}
```

---

module drc.ast.Node;

```
import common;

public import drc.lexer.Token;
public import drc.ast.NodesEnum;

/// Коневой класс всех элементов синтаксического древа Динрус.
abstract class Узел
{
  КатегорияУзла категория; /// Категория данного узла.
  ВидУзла вид; /// Вид данного узла.
  Узел[] отпрыски; // Возможно, будет удалён в будущем.
  Сема* начало, конец; /// Семы в начале и конце данного узла.

  /// Строит объект Узел.
  this(КатегорияУзла категория)
  {
    assert(категория != КатегорияУзла.Неопределённый);
    this.категория = категория;
  }

  проц  установиСемы(Сема* начало, Сема* конец)
  {
    this.начало = начало;
    this.конец = конец;
  }

  Класс устСемы(Класс)(Класс узел)
  {
    узел.установиСемы(this.начало, this.конец);
    return узел;
```

```
  }

  проц  добавьОтпрыск(Узел отпрыск)
  {
    assert(отпрыск !is null, "ошибка в " ~ this.classinfo.имя);
    this.отпрыски ~= отпрыск;
  }

  проц  добавьОпцОтпрыск(Узел отпрыск)
  {
    отпрыск is null || добавьОтпрыск(отпрыск);
  }

  проц  добавьОтпрыски(Узел[] отпрыски)
  {
    assert(отпрыски !is null && delegate{
      foreach (отпрыск; отпрыски)
        if (отпрыск is null)
          return нет;
      return да; }(),
      "ошибка в " ~ this.classinfo.имя
    );
    this.отпрыски ~= отпрыски;
  }

  проц  добавьОпцОтпрыски(Узел[] отпрыски)
  {
    отпрыски is null || добавьОтпрыски(отпрыски);
  }

  /// Возвращает ссылку на Класс, если этот узем может быть в него
преобразован.
  Класс Является(Класс)()
  {
    if (вид == mixin("ВидУзла." ~ Класс.stringof))
      return cast(Класс)cast(ук)this;
    return null;
  }

  /// Преобразует этот узел в Класс.
  Класс в(Класс)()
  {
    return cast(Класс)cast(ук)this;
  }

  /// Возвращает deep-копию данного узла.
  abstract Узел копируй();

  /// Возвращает shallow-копию данного объекта.
  final Узел dup()
  {
    // Выявить размер объекта.
    alias typeof(this.classinfo.иниц[0]) т_байт;
    т_мера размер = this.classinfo.иниц.length;
    // Копировать данные объекта.
    т_байт[] данные = (cast(т_байт*)this)[0..размер].dup;
    return cast(Узел)данные.ptr;
  }

  /// Этот текст внедряется в конструктор класса, наследующего от
  /// Узла. Устанавливается вид члена.
  const ткст установить_вид = `this.вид = mixin("ВидУзла." ~
typeof(this).stringof);`;
```

```
}
```

module drc.ast.NodeCopier;

```
import drc.ast.NodesEnum,
       drc.ast.NodeMembers;

import common;

/// Внедряется в тело класса, наследующего от Узел.
const ткст методКопирования =
  "override typeof(this) копируй()"
  "{"
  "  alias typeof(this) т_этот;"
  "  mixin(генКодКопию(mixin(`ВидУзла.`~т_этот.stringof)));"
  "  return n;"
  "}";

/// Внедряется в тело абстрактного класса БинарноеВыражение.
const ткст бинарноеВыражениеМетодаКопирования =
  "override typeof(this) копируй()"
  "{"
  "  alias typeof(this) т_этот;"
  "  assert(is(ВыражениеЗапятая : БинарноеВыражение), `ВыражениеЗапятая не
наследует от БинарноеВыражение`);"
  "  mixin(генКодКопию(ВидУзла.ВыражениеЗапятая));"
  "  return n;"
  "}";

///  Внедряется в тело абстрактного класса УнарноеВыражение.
const ткст унарноеВыражениеМетодаКопирования =
  "override typeof(this) копируй()"
  "{"
  "  alias typeof(this) т_этот;"
  "  assert(is(ВыражениеАдрес : УнарноеВыражение), `ВыражениеАдрес не
наследует от УнарноеВыражение`);"
  "  mixin(генКодКопию(ВидУзла.ВыражениеАдрес));"
  "  return n;"
  "}";

/// Генерирует рабочий код для копирования предоставленных членов.
private ткст создайКод(ткст[] члены)
{
  ткст[2][] список = разборЧленов(члены);
  ткст код;
  foreach (m; список)
  {
    auto имя = m[0], тип = m[1];
    switch (тип)
    {
    case "": // Copy a член, must not be null.
      // n.член = n.член.копируй();
      код ~= "n."~имя~" = n."~имя~".копируй();"\n;
      break;
    case "?": // Copy a член, may be null.
      // n.член && (n.член = n.член.копируй());
      код ~= "n."~имя~" && (n."~имя~" = n."~имя~".копируй());"\n;
      break;
    case "[]": // Copy an массив of nodes.
      код ~= "n."~имя~" = n."~имя~".dup;"\n // n.член = n.член.dup;
             "foreach (ref x; n."~имя~")"\n  // foreach (ref x; n.член)
             "  x = x.копируй();\n";             //   x = x.копируй();
      break;
```

```
    case "[?]": // Copy an массив of nodes, элементы may be null.
      код ~= "n."~имя~" = n."~имя~".dup;"\n // n.член = n.член.dup;
             "foreach (ref x; n."~имя~")"\n  // foreach (ref x; n.член)
             "  x && (x = x.копируй());\n";      //   x && (x =
x.копируй());
      break;
    case "%": // Copy код verbatim.
      код ~= имя ~ \n;
      break;
    default:
      assert(0, "член неизвестного типа.");
    }
  }
  return код;
}

// pragma(сооб, создайКод(["выр?", "деклы[]", "тип"]));

/// Генерирует код для копирования узла.
ткст генКодКопию(ВидУзла видУзла)
{
  ткст[] m; // Массив копируемых имён членов.

  // Обработка особых случаев.
  if (видУзла == ВидУзла.ТекстовоеВыражение)
    m = ["%n.ткт = n.ткт.dup;"];
  else
    // Поиск членов данного вида узла в таблице.
    m = г_таблицаЧленов[видУзла];

  ткст код =
  // Вначале выполняется shallow-копия.
  "auto n = cast(т_этот)cast(ук)this.dup;\n";

  // Затем копируются члены.
  if (m.length)
    код ~= создайКод(m);

  return код;
}

// pragma(сооб, генКодКопию("ТМассив"));
```

---

module drc.ast.NodeMembers;

```
import drc.ast.NodesEnum;

private alias ВидУзла N;

ткст генТаблицуЧленов()
{ //pragma(сооб, "генТаблицуЧленов()");
  сим[][][] t = [];
  // t.length = г_именаКлассов.length;
  // Setting the length doesn't work in CTFs. Этот is a workaround:
  // FIXME: remove this when dmd issue #2337 has been resolved.
  for (бцел i; i < г_именаКлассов.length; i++)
    t ~= [[]];
  assert(t.length == г_именаКлассов.length);

  t[N.СложнаяДекларация] = ["деклы[]"];
  t[N.ПустаяДекларация] = t[N.НелегальнаяДекларация] =
```

```
  t[N.ДекларацияМодуля] = t[N.ДекларацияИмпорта] = [];
  t[N.ДекларацияАлиаса] = t[N.ДекларацияТипдефа] = ["декл"];
  t[N.ДекларацияПеречня] = ["типОснова?", "члены[]"];
  t[N.ДекларацияЧленаПеречня] = ["тип?", "значение?"];
  t[N.ДекларацияКласса] = t[N.ДекларацияИнтерфейса] = ["основы[]", "деклы?"];
  t[N.ДекларацияСтруктуры] = t[N.ДекларацияСоюза] = ["деклы?"];
  t[N.ДекларацияКонструктора] = ["парамы", "телоФунк"];
  t[N.ДекларацияСтатическогоКонструктора] = t[N.ДекларацияДеструктора] =
  t[N.ДекларацияСтатическогоДеструктора] = t[N.ДекларацияИнварианта] =
  t[N.ДекларацияЮниттеста] = ["телоФунк"];
  t[N.ДекларацияФункции] = ["типВозврата?", "парамы", "телоФунк"];
  t[N.ДекларацияПеременных] = ["узелТипа?", "иниты[?]"];
  t[N.ДекларацияОтладки] = t[N.ДекларацияВерсии] = ["деклы?", "деклыИначе?"];
  t[N.ДекларацияСтатическогоЕсли] = ["условие", "деклыЕсли", "деклыИначе?"];
  t[N.ДекларацияСтатическогоПодтверди] = ["условие", "сообщение?"];
  t[N.ДекларацияШаблона] = ["шпарамы", "констрейнт?", "деклы"];
  t[N.ДекларацияНов] = t[N.ДекларацияУдали] = ["парамы", "телоФунк"];
  t[N.ДекларацияЗащиты] = t[N.ДекларацияКлассаХранения] =
  t[N.ДекларацияКомпоновки] = t[N.ДекларацияРазложи] = ["деклы"];
  t[N.ДекларацияПрагмы] = ["арги[]", "деклы"];
  t[N.ДекларацияСмеси] = ["выражШаблон?", "аргумент?"];
  // Выражения:
  t[N.НелегальноеВыражение] = t[N.ВыражениеИдентификатор] =
  t[N.ВыражениеСпецСема] = t[N.ВыражениеЭтот] =
  t[N.ВыражениеСупер] = t[N.ВыражениеНуль] =
  t[N.ВыражениеДоллар] = t[N.БулевоВыражение] =
  t[N.ЦелВыражение] = t[N.ВыражениеРеал] = t[N.ВыражениеКомплекс] =
  t[N.ВыражениеСим] = t[N.ВыражениеИницПроц] =
  t[N.ВыражениеЛокальногоРазмераАсм] = t[N.ВыражениеАсмРегистр] = [];
  // БинарныеВыражения:
  t[N.ВыражениеУсловия] = ["условие", "лв", "пв"];
  t[N.ВыражениеЗапятая] = t[N.ВыражениеИлиИли] = t[N.ВыражениеИИ] =
  t[N.ВыражениеИли] = t[N.ВыражениеИИли] = t[N.ВыражениеИ] =
  t[N.ВыражениеРавно] = t[N.ВыражениеРавенство] = t[N.ВыражениеОтнош] =
  t[N.ВыражениеВхо] = t[N.ВыражениеЛСдвиг] = t[N.ВыражениеПСдвиг] =
  t[N.ВыражениеБПСдвиг] = t[N.ВыражениеПлюс] = t[N.ВыражениеМинус] =
  t[N.ВыражениеСоедини] = t[N.ВыражениеУмножь] = t[N.ВыражениеДели] =
  t[N.ВыражениеМод] = t[N.ВыражениеПрисвой] = t[N.ВыражениеПрисвойЛСдвиг] =
  t[N.ВыражениеПрисвойПСдвиг] = t[N.ВыражениеПрисвойБПСдвиг] =
  t[N.ВыражениеПрисвойИли] = t[N.ВыражениеПрисвойИ] =
  t[N.ВыражениеПрисвойПлюс] = t[N.ВыражениеПрисвойМинус] =
  t[N.ВыражениеПрисвойДел] = t[N.ВыражениеПрисвойУмн] =
  t[N.ВыражениеПрисвойМод] = t[N.ВыражениеПрисвойИИли] =
  t[N.ВыражениеПрисвойСоед] = t[N.ВыражениеТочка] = ["лв", "пв"];
  // УнарныеВыражения:
  t[N.ВыражениеАдрес] = t[N.ВыражениеПреИнкр] = t[N.ВыражениеПреДекр] =
  t[N.ВыражениеПостИнкр] = t[N.ВыражениеПостДекр] = t[N.ВыражениеДереф] =
  t[N.ВыражениеЗнак] = t[N.ВыражениеНе] = t[N.ВыражениеКомп] =
  t[N.ВыражениеВызов] = t[N.ВыражениеУдали] = t[N.ВыражениеМасштабМодуля] =
  t[N.ВыражениеТипАсм] = t[N.ВыражениеСмещениеАсм] =
  t[N.ВыражениеСегАсм] = ["в"];
  t[N.ВыражениеКаст] = ["тип", "в"];
  t[N.ВыражениеИндекс] = ["в", "арги[]"];
  t[N.ВыражениеСрез] = ["в", "левый?", "правый?"];
  t[N.ВыражениеАсмПослеСкобки] = ["в", "е2"];
  t[N.ВыражениеНов] = ["новАрги[]", "тип", "кторАрги[]"];
  t[N.ВыражениеНовАнонКласс] = ["новАрги[]", "основы[]", "кторАрги[]",
"деклы"];
  t[N.ВыражениеАсмСкобка] = ["в"];
  t[N.ВыражениеЭкземплярШаблона] = ["шарги?"];
  t[N.ВыражениеЛитералМассива] = ["значения[]"];
  t[N.ВыражениеЛитералАМассива] = ["ключи[]", "значения[]"];
  t[N.ВыражениеПодтверди] = ["выр", "сооб?"];
```

```
  t[N.ВыражениеСмесь] = t[N.ВыражениеИмпорта] = ["выр"];
  t[N.ВыражениеТипа] = t[N.ВыражениеИдТипаТочка] =
  t[N.ВыражениеИдТипа] = ["тип"];
  t[N.ВыражениеЯвляется] = ["тип", "типСпец?", "шпарамы?"];
  t[N.ВыражениеЛитералФункции] = ["типВозврата?", "парамы?", "телоФунк"];
  t[N.ВыражениеРодит] = ["следщ"];//paren
  t[N.ВыражениеТрактовки] = ["шарги"];//traits
  t[N.ВыражениеИницМассива] = ["ключи[?]", "значения[]"];
  t[N.ВыражениеИницСтруктуры] = ["значения[]"];
  t[N.ТекстовоеВыражение] = [],
  // Инструкции:
  t[N.НелегальнаяИнструкция] = t[N.ПустаяИнструкция] =
  t[N.ИнструкцияДалее] = t[N.ИнструкцияВсё] =//break
  t[N.ИнструкцияАсмРасклад] = t[N.ИнструкцияНелегальныйАсм] = [];
  t[N.СложнаяИнструкция] = ["инстрции[]"];
  t[N.ИнструкцияТелаФункции] = ["телоФунк?", "телоВхо?", "телоВых?"];
  t[N.ИнструкцияМасштаб] = t[N.ИнструкцияСМеткой] = ["s"];
  t[N.ИнструкцияВыражение] = ["в"];
  t[N.ИнструкцияДекларация] = ["декл"];
  t[N.ИнструкцияЕсли] = ["переменная?", "условие?", "телоЕсли",
"телоИначе?"];
  t[N.ИнструкцияПока] = ["условие", "телоПока"];
  t[N.ИнструкцияДелайПока] = ["телоДелай", "условие"];
  t[N.ИнструкцияПри] = ["иниц?", "условие?", "инкремент?", "телоПри"];//for
  t[N.ИнструкцияСКаждым] = ["парамы", "агрегат", "телоПри"];
  t[N.ИнструкцияДиапазонСКаждым] = ["парамы", "нижний", "верхний",
"телоПри"];
  t[N.ИнструкцияЩит] = ["условие", "телоЩит"];
  t[N.ИнструкцияРеле] = ["значения[]", "телоРеле"];
  t[N.ИнструкцияДефолт] = ["телоДефолта"];
  t[N.ИнструкцияИтог] = ["в?"];
  t[N.ИнструкцияПереход] = ["выгРеле?"];
  t[N.ИнструкцияДля] = ["в", "телоДля"];//with
  t[N.ИнструкцияСинхр] = ["в?", "телоСинхр"];//synchronized
  t[N.ИнструкцияПробуй] = ["телоПробуй", "телаЛови[]", "телоИтожь?"];//try
  t[N.ИнструкцияЛови] = ["парам?", "телоЛови"];//catch
  t[N.ИнструкцияИтожь] = ["телоИтожь"];//finally
  t[N.ИнструкцияСтражМасштаба] = ["телоМасштаба"];
  t[N.ИнструкцияБрось] = ["в"];
  t[N.ИнструкцияЛетучее] = ["телоЛетучего?"];//volatile
  t[N.ИнструкцияБлокАсм] = ["инструкции"];
  t[N.ИнструкцияАсм] = ["операнды[]"];
  t[N.ИнструкцияПрагма] = ["арги[]", "телоПрагмы"];
  t[N.ИнструкцияСмесь] = ["выражШаблон"];
  t[N.ИнструкцияСтатическоеЕсли] = ["условие", "телоЕсли", "телоИначе?"];
  t[N.ИнструкцияСтатическоеПодтверди] = ["условие", "сообщение?"];
  t[N.ИнструкцияОтладка] = t[N.ИнструкцияВерсия] = ["телоГлавного",
"телоИначе?"];
  // УзлыТипов:
  t[N.НелегальныйТип] = t[N.ИнтегральныйТип] =
  t[N.ТМасштабМодуля] = t[N.ТИдентификатор] = [];
  t[N.КвалифицированныйТип] = ["лв", "пв"];
  t[N.ТТип] = ["в"];
  t[N.ТЭкземплярШаблона] = ["шарги?"];
  t[N.ТМассив] = ["следщ", "ассоцТип?", "е1?", "е2?"];
  t[N.ТФункция] = t[N.ТДелегат] = ["типВозврата", "парамы"];
  t[N.ТУказательНаФункСи] = ["следщ", "парамы?"];
  t[N.ТУказатель] = t[N.ТипКлассОснова] =
  t[N.ТКонст] = t[N.ТИнвариант] = ["следщ"];
  // Параметры:
  t[N.Параметр] = ["тип?", "дефЗначение?"];
  t[N.Параметры] = t[N.ПараметрыШаблона] =
  t[N.АргументыШаблона] = ["отпрыски[]"];
```

```
  t[N.ПараметрАлиасШаблона] = t[N.ПараметрТипаШаблона] =
  t[N.ПараметрЭтотШаблона] = ["типСпец?", "дефТип?"];
  t[N.ПараметрШаблонЗначения] = ["типЗначение", "спецЗначение?",
"дефЗначение?"];
  t[N.ПараметрКортежШаблона] = [];

  ткст код = "[";
  // Iterate over the elements in the таблица and create an массив.
  foreach (m; t)
  {
    if (!m.length) {
      код ~= "[],";
      continue; // No члены, добавь "[]," and continue.
    }
    код ~= '[';
    foreach (n; m)
      код ~= `"` ~ n ~ `",`;
    код[код.length-1] = ']'; // Overwrite last comma.
    код ~= ',';
  }
  код[код.length-1] = ']'; // Overwrite last comma.
  return код;
}

/// Таблица-листинг подузлов всех классов, унаследованных от Узел.
static const сим[][][/+ВидУзла.max+1+/] г_таблицаЧленов =
mixin(генТаблицуЧленов());

/// Вспомогательная функция, парсирующая спецтексты в г_таблицаЧленов.
///
/// Основной синтаксис:
/// $(PRE
/// Член := Массив | Массив2 | ОпционныйУзел | Узел | Код
/// Массив := Идентификатор "[]"
/// Массив2 := Идентификатор "[?]"
/// ОпционныйУзел := Идентификатор "?"
/// Узел := Идентификатор
/// Код  := "%" ЛюбойСим*
/// $(MODLINK2 drc.lexer.Identifier, Идентификатор)
/// )
/// Параметры:
///   члены = парсируемые члены-тексты.
/// Возвращает:
///   массив кортежей (Имя, Тип), где Имя - точное имя члена
///   а Тип может иметь одно из следующих значений: "[]", "[?]", "?", "" или
"%".
сим[][2][] разборЧленов(сим[][] члены)
{
  сим[][2][] результат;
  foreach (член; члены)
    if (член.length > 2 && член[$-2..$] == "[]")
      результат ~= [член[0..$-2], "[]"]; // Strip off trailing '[]'.
    else if (член.length > 3 && член[$-3..$] == "[?]")
      результат ~= [член[0..$-3], "[?]"]; // Strip off trailing '[?]'.
    else if (член[$-1] == '?')
      результат ~= [член[0..$-1], "?"]; // Strip off trailing '?'.
    else if (член[0] == '%')
      результат ~= [член[1..$], "%"]; // Strip off preceding '%'.
    else
      результат ~= [член, ""]; // Nothing в тктip off.
  return результат;
}
```

```
module drc.ast.NodesEnum;

/// Перечисляет категории узла.
enum КатегорияУзла : бкрат
{
  Неопределённый,
  Декларация,
  Инструкция,
  Выражение,
  Тип,
  Иное // Параметр
}

/// Список имен классов, наследующих от Узел.
static const сим[][] г_именаКлассов = [
  // Declarations:
  "СложнаяДекларация",
  "ПустаяДекларация",
  "НелегальнаяДекларация",
  "ДекларацияМодуля",
  "ДекларацияИмпорта",
  "ДекларацияАлиаса",
  "ДекларацияТипдефа",
  "ДекларацияПеречня",
  "ДекларацияЧленаПеречня",
  "ДекларацияКласса",
  "ДекларацияИнтерфейса",
  "ДекларацияСтруктуры",
  "ДекларацияСоюза",
  "ДекларацияКонструктора",
  "ДекларацияСтатическогоКонструктора",
  "ДекларацияДеструктора",
  "ДекларацияСтатическогоДеструктора",
  "ДекларацияФункции",
  "ДекларацияПеременных",
  "ДекларацияИнварианта",
  "ДекларацияЮниттеста",
  "ДекларацияОтладки",
  "ДекларацияВерсии",
  "ДекларацияСтатическогоЕсли",
  "ДекларацияСтатическогоПодтверди",
  "ДекларацияШаблона",
  "ДекларацияНов",
  "ДекларацияУдали",
  "ДекларацияЗащиты",
  "ДекларацияКлассаХранения",
  "ДекларацияКомпоновки",
  "ДекларацияРазложи",
  "ДекларацияПрагмы",
  "ДекларацияСмеси",

  // Statements:
  "СложнаяИнструкция",
  "НелегальнаяИнструкция",
  "ПустаяИнструкция",
  "ИнструкцияТелаФункции",
  "ИнструкцияМасштаб",
  "ИнструкцияСМеткой",
  "ИнструкцияВыражение",
  "ИнструкцияДекларация",
  "ИнструкцияЕсли",
  "ИнструкцияПока",
```

```
"ИнструкцияДелайПока",
"ИнструкцияПри",
"ИнструкцияСКаждым",
"ИнструкцияДиапазонСКаждым", // D2.0
"ИнструкцияЩит",
"ИнструкцияРеле",
"ИнструкцияДефолт",
"ИнструкцияДалее",
"ИнструкцияВсё",
"ИнструкцияИтог",
"ИнструкцияПереход",
"ИнструкцияДля",
"ИнструкцияСинхр",
"ИнструкцияПробуй",
"ИнструкцияЛови",
"ИнструкцияИтожь",
"ИнструкцияСтражМасштаба",
"ИнструкцияБрось",
"ИнструкцияЛетучее",
"ИнструкцияБлокАсм",
"ИнструкцияАсм",
"ИнструкцияАсмРасклад",
"ИнструкцияНелегальныйАсм",
"ИнструкцияПрагма",
"ИнструкцияСмесь",
"ИнструкцияСтатическоеЕсли",
"ИнструкцияСтатическоеПодтверди",
"ИнструкцияОтладка",
"ИнструкцияВерсия",

// Expressions:
"НелегальноеВыражение",
"ВыражениеУсловия",
"ВыражениеЗапятая",
"ВыражениеИлиИли",
"ВыражениеИИ",
"ВыражениеИли",
"ВыражениеИИли",
"ВыражениеИ",
"ВыражениеРавно",
"ВыражениеРавенство",
"ВыражениеОтнош",
"ВыражениеВхо",
"ВыражениеЛСдвиг",
"ВыражениеПСдвиг",
"ВыражениеБПСдвиг",
"ВыражениеПлюс",
"ВыражениеМинус",
"ВыражениеСоедини",
"ВыражениеУмножь",
"ВыражениеДели",
"ВыражениеМод",
"ВыражениеПрисвой",
"ВыражениеПрисвойЛСдвиг",
"ВыражениеПрисвойПСдвиг",
"ВыражениеПрисвойБПСдвиг",
"ВыражениеПрисвойИли",
"ВыражениеПрисвойИ",
"ВыражениеПрисвойПлюс",
"ВыражениеПрисвойМинус",
"ВыражениеПрисвойДел",
"ВыражениеПрисвойУмн",
"ВыражениеПрисвойМод",
```

```
"ВыражениеПрисвойИИли",
"ВыражениеПрисвойСоед",
"ВыражениеАдрес",
"ВыражениеПреИнкр",
"ВыражениеПреДекр",
"ВыражениеПостИнкр",
"ВыражениеПостДекр",
"ВыражениеДереф",
"ВыражениеЗнак",
"ВыражениеНе",
"ВыражениеКомп",
"ВыражениеВызов",
"ВыражениеНов",
"ВыражениеНовАнонКласс",
"ВыражениеУдали",
"ВыражениеКаст",
"ВыражениеИндекс",
"ВыражениеСрез",
"ВыражениеМасштабМодуля",
"ВыражениеИдентификатор",
"ВыражениеСпецСема",
"ВыражениеТочка",
"ВыражениеЭкземплярШаблона",
"ВыражениеЭтот",
"ВыражениеСупер",
"ВыражениеНуль",
"ВыражениеДоллар",
"БулевоВыражение",
"ЦелВыражение",
"ВыражениеРеал",
"ВыражениеКомплекс",
"ВыражениеСим",
"ТекстовоеВыражение",
"ВыражениеЛитералМассива",
"ВыражениеЛитералАМассива",
"ВыражениеПодтверди",
"ВыражениеСмесь",
"ВыражениеИмпорта",
"ВыражениеТипа",
"ВыражениеИдТипаТочка",
"ВыражениеИдТипа",
"ВыражениеЯвляется",
"ВыражениеРодит",
"ВыражениеЛитералФункции",
"ВыражениеТрактовки", // D2.0
"ВыражениеИницПроц",
"ВыражениеИницМассива",
"ВыражениеИницСтруктуры",
"ВыражениеТипАсм",
"ВыражениеСмещениеАсм",
"ВыражениеСегАсм",
"ВыражениеАсмПослеСкобки",
"ВыражениеАсмСкобка",
"ВыражениеЛокальногоРазмераАсм",
"ВыражениеАсмРегистр",

// Типы:
"НелегальныйТип",
"ИнтегральныйТип",
"КвалифицированныйТип",
"ТМасштабМодуля",
"ТИдентификатор",
"ТТип",
```

```
  "ТЭкземплярШаблона",
  "ТУказатель",
  "ТМассив",
  "ТФункция",
  "ТДелегат",
  "ТУказательНаФункСи",
  "ТипКлассОснова",
  "ТКонст", // D2.0
  "ТИнвариант", // D2.0

  // Параметры:
  "Параметр",
  "Параметры",
  "ПараметрАлиасШаблона",
  "ПараметрТипаШаблона",
  "ПараметрЭтотШаблона", // D2.0
  "ПараметрШаблонЗначения",
  "ПараметрКортежШаблона",
  "ПараметрыШаблона",
  "АргументыШаблона",
];

/// Генерирует члены перечня ВидУзла.
ткст генерируйЧленыВидовУзла()
{
  ткст текст;
  foreach (имяКласса; г_именаКлассов)
    текст ~= имяКласса ~ ",";
  return текст;
}
// pragma(сооб, генерируйЧленыВидовУзла());

version(DDoc)
  /// Вид узла идентифицирует каждый класс,наследующий от Узел.
  enum ВидУзла : бкрат;
else
mixin(
  "enum ВидУзла : бкрат"
  "{"
    ~ генерируйЧленыВидовУзла ~
  "}"
);
```

---

module drc.ast.Parameters;

```
import drc.ast.Node,
       drc.ast.Type,
       drc.ast.Expression,
       drc.ast.NodeCopier;
import drc.lexer.Identifier;
import drc.Enums;

/// Функция или параметр foreach.
class Параметр : Узел
{
  КлассХранения кхр; /// Классы хранения параметра.
  УзелТипа тип; /// Тип параметра.
  Идентификатор* имя; /// Имя параметра.
  Выражение дефЗначение; /// Дефолтное значение инициализации.
```

```
  this(КлассХранения кхр, УзелТипа тип, Идентификатор* имя, Выражение
дефЗначение);

  /// Возвращает да, если из_ a D-style variadic parameter.
  /// E.g.: func(цел[] значения ...)
  бул ДиВариадический_ли();

  /// Возвращает да, если из_ a C-style variadic parameter.
  /// E.g.: func(...)
  бул СиВариадический_ли();

  /// Возвращает да, если из_ a D- or C-style variadic parameter.
  бул вариадический_ли();

  /// Returns да if this parameter is lazy.
  бул лэйзи_ли();

  mixin(методКопирования);
}

/// Массив параметров.
class Параметры : Узел
{
  this();

  бул естьВариадические();

  бул естьЛэйзи();

  проц  opCatAssign(Параметр парам);

  Параметр[] элементы();

  т_мера length();

  mixin(методКопирования);
}

/*~~~~~~~~~~~~~~~~~~~~~
~ Шаблон параметры: ~
~~~~~~~~~~~~~~~~~~~~~*/

/// Абстрактный класс-основа для всех параметров шаблонов.
abstract class ПараметрШаблона : Узел
{
  Идентификатор* идент;
  this(Идентификатор* идент);
}

/// E.g.: (alias T)
class ПараметрАлиасШаблона : ПараметрШаблона
{
  УзелТипа типСпец, дефТип;
  this(Идентификатор* идент, УзелТипа типСпец, УзелТипа дефТип);
  mixin(методКопирования);
}

/// E.g.: (T t)
class ПараметрТипаШаблона : ПараметрШаблона
{
  УзелТипа типСпец, дефТип;
  this(Идентификатор* идент, УзелТипа типСпец, УзелТипа дефТип);
  mixin(методКопирования);
```

```
}

// version(D2)
// {
/// E.g.: (this T)
class ПараметрЭтотШаблона : ПараметрШаблона
{
  УзелТипа типСпец, дефТип;
  this(Идентификатор* идент, УзелТипа типСпец, УзелТипа дефТип);
  mixin(методКопирования);
}
// }

/// E.g.: (T)
class ПараметрШаблонЗначения : ПараметрШаблона
{
  УзелТипа типЗначение;
  Выражение спецЗначение, дефЗначение;
  this(УзелТипа типЗначение, Идентификатор* идент, Выражение спецЗначение,
Выражение дефЗначение);
  mixin(методКопирования);
}

/// E.g.: (T...)
class ПараметрКортежШаблона : ПараметрШаблона
{
  this(Идентификатор* идент);
  mixin(методКопирования);
}

/// Массив параметров шаблона.
class ПараметрыШаблона : Узел
{
  this();

  проц  opCatAssign(ПараметрШаблона параметр);

  ПараметрШаблона[] элементы();

  mixin(методКопирования);
}

/// Массив аргументов шаблона.
class АргументыШаблона : Узел
{
  this();

  проц  opCatAssign(Узел аргумент);

  mixin(методКопирования);
}
```

module drc.ast.Statement;

```
import drc.ast.Node;

/// The корень class of all инструкции.
abstract class Инструкция : Узел
{
  this()
  {
    super(КатегорияУзла.Инструкция);
  }
```

```
  override abstract Инструкция копируй();
}
```

module drc.ast.Statements;

```
public import drc.ast.Statement;
import drc.ast.Node,
       drc.ast.Expression,
       drc.ast.Declaration,
       drc.ast.Type,
       drc.ast.Parameters,
       drc.ast.NodeCopier;
import drc.lexer.IdTable;

class СложнаяИнструкция : Инструкция
{
  this()
  {
    mixin(установить_вид);
  }

  проц  opCatAssign(Инструкция s)
  {
    добавьОтпрыск(s);
  }

  Инструкция[] инстрции()
  {
    return cast(Инструкция[])this.отпрыски;
  }

  проц  инстрции(Инструкция[] инстрции)
  {
    this.отпрыски = инстрции;
  }

  mixin(методКопирования);
}

class НелегальнаяИнструкция : Инструкция
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

class ПустаяИнструкция : Инструкция
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

class ИнструкцияТелаФункции : Инструкция
{
  Инструкция телоФунк, телоВхо, телоВых;
  Идентификатор* outIdent;
```

```
  this()
  {
    mixin(установить_вид);
  }

  проц  завершиКонструкцию()
  {
    добавьОпцОтпрыск(телоФунк);
    добавьОпцОтпрыск(телоВхо);
    добавьОпцОтпрыск(телоВых);
  }

  бул пуст_ли()
  {
    return телоФунк is null;
  }

  mixin(методКопирования);
}

class ИнструкцияМасштаб : Инструкция
{
  Инструкция s;
  this(Инструкция s)
  {
    mixin(установить_вид);
    добавьОтпрыск(s);
    this.s = s;
  }
  mixin(методКопирования);
}

class ИнструкцияСМеткой : Инструкция
{
  Идентификатор* лейбл;
  Инструкция s;
  this(Идентификатор* лейбл, Инструкция s)
  {
    mixin(установить_вид);
    добавьОтпрыск(s);
    this.лейбл = лейбл;
    this.s = s;
  }
  mixin(методКопирования);
}

class ИнструкцияВыражение : Инструкция
{
  Выражение в;
  this(Выражение в)
  {
    mixin(установить_вид);
    добавьОтпрыск(в);
    this.в = в;
  }
  mixin(методКопирования);
}

class ИнструкцияДекларация : Инструкция
{
  Декларация декл;
  this(Декларация декл)
  {
```

```
    mixin(установить_вид);
    добавьОтпрыск(декл);
    this.декл = декл;
  }
  mixin(методКопирования);
}

class ИнструкцияЕсли : Инструкция
{
  Инструкция переменная; // ДекларацияАвто or ДекларацияПеременной
  Выражение условие;
  Инструкция телоЕсли;
  Инструкция телоИначе;
  this(Инструкция переменная, Выражение условие, Инструкция телоЕсли,
Инструкция телоИначе)
  {
    mixin(установить_вид);
    if (переменная)
      добавьОтпрыск(переменная);
    else
      добавьОтпрыск(условие);
    добавьОтпрыск(телоЕсли);
    добавьОпцОтпрыск(телоИначе);

    this.переменная = переменная;
    this.условие = условие;
    this.телоЕсли = телоЕсли;
    this.телоИначе = телоИначе;
  }
  mixin(методКопирования);
}

class ИнструкцияПока : Инструкция
{
  Выражение условие;
  Инструкция телоПока;
  this(Выражение условие, Инструкция телоПока)
  {
    mixin(установить_вид);
    добавьОтпрыск(условие);
    добавьОтпрыск(телоПока);

    this.условие = условие;
    this.телоПока = телоПока;
  }
  mixin(методКопирования);
}

class ИнструкцияДелайПока : Инструкция
{
  Инструкция телоДелай;
  Выражение условие;
  this(Выражение условие, Инструкция телоДелай)
  {
    mixin(установить_вид);
    добавьОтпрыск(телоДелай);
    добавьОтпрыск(условие);

    this.условие = условие;
    this.телоДелай = телоДелай;
  }
  mixin(методКопирования);
}
```

```
class ИнструкцияПри : Инструкция
{
  Инструкция иниц;
  Выражение условие, инкремент;
  Инструкция телоПри;

  this(Инструкция иниц, Выражение условие, Выражение инкремент, Инструкция
телоПри)
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(иниц);
    добавьОпцОтпрыск(условие);
    добавьОпцОтпрыск(инкремент);
    добавьОтпрыск(телоПри);

    this.иниц = иниц;
    this.условие = условие;
    this.инкремент = инкремент;
    this.телоПри = телоПри;
  }
  mixin(методКопирования);
}

class ИнструкцияСКаждым : Инструкция
{
  ТОК лекс;
  Параметры парамы;
  Выражение агрегат;
  Инструкция телоПри;

  this(ТОК лекс, Параметры парамы, Выражение агрегат, Инструкция телоПри)
  {
    mixin(установить_вид);
    добавьОтпрыски([cast(Узел)парамы, агрегат, телоПри]);

    this.лекс = лекс;
    this.парамы = парамы;
    this.агрегат = агрегат;
    this.телоПри = телоПри;
  }

  /// Возвращает да, если из_ a foreach_reverse statement.
  бул isReverse()
  {
    return лекс == ТОК.Длявсех_реверс;
  }

  mixin(методКопирования);
}

// version(D2)
// {
class ИнструкцияДиапазонСКаждым : Инструкция
{
  ТОК лекс;
  Параметры парамы;
  Выражение нижний, верхний;
  Инструкция телоПри;

  this(ТОК лекс, Параметры парамы, Выражение нижний, Выражение верхний,
Инструкция телоПри)
  {
```

```
    mixin(установить_вид);
    добавьОтпрыски([cast(Узел)парамы, нижний, верхний, телоПри]);

    this.лекс = лекс;
    this.парамы = парамы;
    this.нижний = нижний;
    this.верхний = верхний;
    this.телоПри = телоПри;
  }
  mixin(методКопирования);
}
// }

class ИнструкцияЩит : Инструкция
{
  Выражение условие;
  Инструкция телоЩит;

  this(Выражение условие, Инструкция телоЩит)
  {
    mixin(установить_вид);
    добавьОтпрыск(условие);
    добавьОтпрыск(телоЩит);

    this.условие = условие;
    this.телоЩит = телоЩит;
  }
  mixin(методКопирования);
}

class ИнструкцияРеле : Инструкция
{
  Выражение[] значения;
  Инструкция телоРеле;

  this(Выражение[] значения, Инструкция телоРеле)
  {
    mixin(установить_вид);
    добавьОтпрыски(значения);
    добавьОтпрыск(телоРеле);

    this.значения = значения;
    this.телоРеле = телоРеле;
  }
  mixin(методКопирования);
}

class ИнструкцияДефолт : Инструкция
{
  Инструкция телоДефолта;
  this(Инструкция телоДефолта)
  {
    mixin(установить_вид);
    добавьОтпрыск(телоДефолта);

    this.телоДефолта = телоДефолта;
  }
  mixin(методКопирования);
}

class ИнструкцияДалее : Инструкция
{
  Идентификатор* идент;
```

```
  this(Идентификатор* идент)
  {
    mixin(установить_вид);
    this.идент = идент;
  }
  mixin(методКопирования);
}

class ИнструкцияВсё : Инструкция
{
  Идентификатор* идент;
  this(Идентификатор* идент)
  {
    mixin(установить_вид);
    this.идент = идент;
  }
  mixin(методКопирования);
}

class ИнструкцияИтог : Инструкция
{
  Выражение в;
  this(Выражение в)
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(в);
    this.в = в;
  }
  mixin(методКопирования);
}

class ИнструкцияПереход : Инструкция
{
  Идентификатор* идент;
  Выражение выРеле;
  this(Идентификатор* идент, Выражение выРеле)
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(выРеле);
    this.идент = идент;
    this.выРеле = выРеле;
  }
  mixin(методКопирования);
}

class ИнструкцияДля : Инструкция
{
  Выражение в;
  Инструкция телоДля;
  this(Выражение в, Инструкция телоДля)
  {
    mixin(установить_вид);
    добавьОтпрыск(в);
    добавьОтпрыск(телоДля);

    this.в = в;
    this.телоДля = телоДля;
  }
  mixin(методКопирования);
}

class ИнструкцияСинхр : Инструкция
{
```

```
  Выражение в;
  Инструкция телоСинхр;
  this(Выражение в, Инструкция телоСинхр)
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(в);
    добавьОтпрыск(телоСинхр);

    this.в = в;
    this.телоСинхр = телоСинхр;
  }
  mixin(методКопирования);
}

class ИнструкцияПробуй : Инструкция
{
  Инструкция телоПробуй;
  ИнструкцияЛови[] телаЛови;
  ИнструкцияИтожь телоИтожь;
  this(Инструкция телоПробуй, ИнструкцияЛови[] телаЛови, ИнструкцияИтожь
телоИтожь)
  {
    mixin(установить_вид);
    добавьОтпрыск(телоПробуй);
    добавьОпцОтпрыски(телаЛови);
    добавьОпцОтпрыск(телоИтожь);

    this.телоПробуй = телоПробуй;
    this.телаЛови = телаЛови;
    this.телоИтожь = телоИтожь;
  }
  mixin(методКопирования);
}

class ИнструкцияЛови : Инструкция
{
  Параметр парам;
  Инструкция телоЛови;
  this(Параметр парам, Инструкция телоЛови)
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(парам);
    добавьОтпрыск(телоЛови);
    this.парам = парам;
    this.телоЛови = телоЛови;
  }
  mixin(методКопирования);
}

class ИнструкцияИтожь : Инструкция
{
  Инструкция телоИтожь;
  this(Инструкция телоИтожь)
  {
    mixin(установить_вид);
    добавьОтпрыск(телоИтожь);
    this.телоИтожь = телоИтожь;
  }
  mixin(методКопирования);
}

class ИнструкцияСтражМасштаба : Инструкция
{
```

```
  Идентификатор* условие;
  Инструкция телоМасштаба;
  this(Идентификатор* условие, Инструкция телоМасштаба)
  {
    mixin(установить_вид);
    добавьОтпрыск(телоМасштаба);
    this.условие = условие;
    this.телоМасштаба = телоМасштаба;
  }
  mixin(методКопирования);
}

class ИнструкцияБрось : Инструкция
{
  Выражение в;
  this(Выражение в)
  {
    mixin(установить_вид);
    добавьОтпрыск(в);
    this.в = в;
  }
  mixin(методКопирования);
}

class ИнструкцияЛетучее : Инструкция
{
  Инструкция телоЛетучего;
  this(Инструкция телоЛетучего)
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(телоЛетучего);
    this.телоЛетучего = телоЛетучего;
  }
  mixin(методКопирования);
}

class ИнструкцияБлокАсм : Инструкция
{
  СложнаяИнструкция инструкции;
  this(СложнаяИнструкция инструкции)
  {
    mixin(установить_вид);
    добавьОтпрыск(инструкции);
    this.инструкции = инструкции;
  }
  mixin(методКопирования);
}

class ИнструкцияАсм : Инструкция
{
  Идентификатор* идент;
  Выражение[] операнды;
  this(Идентификатор* идент, Выражение[] операнды)
  {
    mixin(установить_вид);
    добавьОпцОтпрыски(операнды);
    this.идент = идент;
    this.операнды = операнды;
  }
  mixin(методКопирования);
}

class ИнструкцияАсмРасклад : Инструкция
```

```
{
  цел число;
  this(цел число)
  {
    mixin(установить_вид);
    this.число = число;
  }
  mixin(методКопирования);
}

class ИнструкцияНелегальныйАсм : НелегальнаяИнструкция
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

class ИнструкцияПрагма : Инструкция
{
  Идентификатор* идент;
  Выражение[] арги;
  Инструкция телоПрагмы;
  this(Идентификатор* идент, Выражение[] арги, Инструкция телоПрагмы)
  {
    mixin(установить_вид);
    добавьОпцОтпрыски(арги);
    добавьОтпрыск(телоПрагмы);

    this.идент = идент;
    this.арги = арги;
    this.телоПрагмы = телоПрагмы;
  }
  mixin(методКопирования);
}

class ИнструкцияСмесь : Инструкция
{
  Выражение выражШаблон;
  Идентификатор* идентСмеси;
  this(Выражение выражШаблон, Идентификатор* идентСмеси)
  {
    mixin(установить_вид);
    добавьОтпрыск(выражШаблон);
    this.выражШаблон = выражШаблон;
    this.идентСмеси = идентСмеси;
  }
  mixin(методКопирования);
}

class ИнструкцияСтатическоеЕсли : Инструкция
{
  Выражение условие;
  Инструкция телоЕсли, телоИначе;
  this(Выражение условие, Инструкция телоЕсли, Инструкция телоИначе)
  {
    mixin(установить_вид);
    добавьОтпрыск(условие);
    добавьОтпрыск(телоЕсли);
    добавьОпцОтпрыск(телоИначе);
    this.условие = условие;
    this.телоЕсли = телоЕсли;
```

```
    this.телоИначе = телоИначе;
  }
  mixin(методКопирования);
}

class ИнструкцияСтатическоеПодтверди : Инструкция
{
  Выражение условие, сообщение;
  this(Выражение условие, Выражение сообщение)
  {
    mixin(установить_вид);
    добавьОтпрыск(условие);
    добавьОпцОтпрыск(сообщение);
    this.условие = условие;
    this.сообщение = сообщение;
  }
  mixin(методКопирования);
}

abstract class ИнструкцияУсловнойКомпиляции : Инструкция
{
  Сема* услов;
  Инструкция телоГлавного, телоИначе;
  this(Сема* услов, Инструкция телоГлавного, Инструкция телоИначе)
  {
    добавьОтпрыск(телоГлавного);
    добавьОпцОтпрыск(телоИначе);
    this.услов = услов;
    this.телоГлавного = телоГлавного;
    this.телоИначе = телоИначе;
  }
}

class ИнструкцияОтладка : ИнструкцияУсловнойКомпиляции
{
  this(Сема* услов, Инструкция телоОтладки, Инструкция телоИначе)
  {
    super(услов, телоОтладки, телоИначе);
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

class ИнструкцияВерсия : ИнструкцияУсловнойКомпиляции
{
  this(Сема* услов, Инструкция телоВерсии, Инструкция телоИначе)
  {
    super(услов, телоВерсии, телоИначе);
    mixin(установить_вид);
  }
  mixin(методКопирования);
}
```

---

module drc.ast.Type;

```
import drc.ast.Node;
import drc.semantic.Types,
       drc.semantic.Symbol;

/// Корневой класс узлов всех типов.
abstract class УзелТипа : Узел
```

```
{
  УзелТипа следщ; /// Следующий тип в цепочке типов.
  Тип тип; /// Семантический тип данного узлового типа.
  Символ символ;

  this()
  {
    this(null);
  }

  this(УзелТипа следщ)
  {
    super(КатегорияУзла.Тип);
    добавьОпцОтпрыск(следщ);
    this.следщ = следщ;
  }

  /// Возвращает корневой тип цепочки типов.
  УзелТипа типОснова()
  {
    auto тип = this;
    while (тип.следщ)
      тип = тип.следщ;
    return тип;
  }

  /// Возвращает да, если член 'тип' не null.
  бул естьТип()
  {
    return тип !is null;
  }

  /// Возвращает да, если член 'символ' не null.
  бул естьСимвол()
  {
    return символ !is null;
  }

  override abstract УзелТипа копируй();
}
```

---

module drc.ast.Types;

```
public import drc.ast.Type;
import drc.ast.Node,
       drc.ast.Expression,
       drc.ast.Parameters,
       drc.ast.NodeCopier;
import drc.lexer.Identifier;
import drc.semantic.Types;
import drc.Enums;

/// Синтаксис ошибка.
class НелегальныйТип : УзелТипа
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}
```

```
/// сим, цел, плав etc.
class ИнтегральныйТип : УзелТипа
{
  ТОК лекс;
  this(ТОК лекс)
  {
    mixin(установить_вид);
    this.лекс = лекс;
  }
  mixin(методКопирования);
}

/// Идентификатор
class ТИдентификатор : УзелТипа
{
  Идентификатор* идент;
  this(Идентификатор* идент)
  {
    mixin(установить_вид);
    this.идент = идент;
  }
  mixin(методКопирования);
}

/// Тип "." Тип
class КвалифицированныйТип : УзелТипа
{
  alias следщ лв; /// Left-hand сторона тип.
  УзелТипа пв; /// Right-hand сторона тип.
  this(УзелТипа лв, УзелТипа пв)
  {
    super(лв);
    mixin(установить_вид);
    добавьОтпрыск(пв);
    this.пв = пв;
  }
  mixin(методКопирования);
}

/// "." Тип
class ТМасштабМодуля : УзелТипа
{
  this()
  {
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

/// "typeof" "(" Выражение ")" or$(BR)
/// "typeof" "(" "return" ")" (D2.0)
class ТТип : УзелТипа
{
  Выражение в;
  /// "typeof" "(" Выражение ")"
  this(Выражение в)
  {
    this();
    добавьОтпрыск(в);
    this.в = в;
  }
```

```
  /// При D2.0: "typeof" "(" "return" ")"
  this()
  {
    mixin(установить_вид);
  }

  /// Возвращает да, если из_ а "typeof(return)".
  бул типВозврата_ли()
  {
    return в is null;
  }

  mixin(методКопирования);
}

/// Идентификатор "!" "(" ПараметрыШаблона? ")"
class ТЭкземплярШаблона : УзелТипа
{
  Идентификатор* идент;
  АргументыШаблона шарги;
  this(Идентификатор* идент, АргументыШаблона шарги)
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(шарги);
    this.идент = идент;
    this.шарги = шарги;
  }
  mixin(методКопирования);
}

/// Тип *
class ТУказатель : УзелТипа
{
  this(УзелТипа следщ)
  {
    super(следщ);
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

/// Dynamic массив: T[] or$(BR)
/// Статический массив: T[E] or$(BR)
/// Срез массив (for tuples): T[E..E] or$(BR)
/// Associative массив: T[T]
class ТМассив : УзелТипа
{
  Выражение e1, e2;
  УзелТипа ассоцТип;

  this(УзелТипа t)
  {
    super(t);
    mixin(установить_вид);
  }

  this(УзелТипа t, Выражение e1, Выражение e2)
  {
    this(t);
    добавьОтпрыск(e1);
    добавьОпцОтпрыск(e2);
    this.e1 = e1;
    this.e2 = e2;
```

```
  }

  this(УзелТипа t, УзелТипа ассоцТип)
  {
    this(t);
    добавьОтпрыск(ассоцТип);
    this.ассоцТип = ассоцТип;
  }

  бул динамический_ли()
  {
    return !ассоцТип && !e1;
  }

  бул статический_ли()
  {
    return e1 && !e2;
  }

  бул срез_ли()
  {
    return e1 && e2;
  }

  бул ассоциативный_ли()
  {
    return ассоцТип !is null;
  }

  mixin(методКопирования);
}

/// ТипИтога "function" "(" Параметры? ")"
class ТФункция : УзелТипа
{
  alias следщ типВозврата;
  Параметры парамы;
  this(УзелТипа типВозврата, Параметры парамы)
  {
    super(типВозврата);
    mixin(установить_вид);
    добавьОтпрыск(парамы);
    this.парамы = парамы;
  }
  mixin(методКопирования);
}

/// ТипИтога "delegate" "(" Параметры? ")"
class ТДелегат : УзелТипа
{
  alias следщ типВозврата;
  Параметры парамы;
  this(УзелТипа типВозврата, Параметры парамы)
  {
    super(типВозврата);
    mixin(установить_вид);
    добавьОтпрыск(парамы);
    this.парамы = парамы;
  }
  mixin(методКопирования);
}

/// Тип "(" BasicType2 Идентификатор ")" "(" Параметры? ")"
```

```
class ТУказательНаФункСи : УзелТипа
{
  Параметры парамы;
  this(УзелТипа тип, Параметры парамы)
  {
    super(тип);
    mixin(установить_вид);
    добавьОпцОтпрыск(парамы);
  }
  mixin(методКопирования);
}

/// "class" Идентификатор : BaseClasses
class ТипКлассОснова : УзелТипа
{
  Защита защ;
  this(Защита защ, УзелТипа тип)
  {
    super(тип);
    mixin(установить_вид);
    this.защ = защ;
  }
  mixin(методКопирования);
}

// version(D2)
// {
/// "const" "(" Тип ")"
class ТКонст : УзелТипа
{
  this(УзелТипа следщ)
  {
    // Если t is null: cast(const)
    super(следщ);
    mixin(установить_вид);
  }
  mixin(методКопирования);
}

/// "invariant" "(" Тип ")"
class ТИнвариант : УзелТипа
{
  this(УзелТипа следщ)
  {
    // Если t is null: cast(invariant)
    super(следщ);
    mixin(установить_вид);
  }
  mixin(методКопирования);
}
// } // version(D2)
```

module drc.ast.Visitor;

```
import drc.ast.Node,
       drc.ast.Declarations,
       drc.ast.Expressions,
       drc.ast.Statements,
       drc.ast.Types,
       drc.ast.Parameters;

/// Генерирует методы визита.
```

```
///
/// Напр.:
/// ---
/// Декларация посети(ДекларацияКласса){return null;};
/// Выражение посети(ВыражениеЗапятая){return null;};
/// ---
ткст генерируйМетодыВизита()
{
  ткст текст;
  foreach (имяКласса; г_именаКлассов)
    текст ~= "типВозврата!(\""~имяКласса~"\") посети("~имяКласса~"
узел){return узел;}\n";
  return текст;
}
// pragma(сооб, generateAbтктactVisitMethods());

/// Получает соответствующий тип возврата для предложенного класса.
template типВозврата(ткст имяКласса)
{
  static if (is(typeof(mixin(имяКласса)) : Декларация))
    alias Декларация типВозврата;
  else
  static if (is(typeof(mixin(имяКласса)) : Инструкция))
    alias Инструкция типВозврата;
  else
  static if (is(typeof(mixin(имяКласса)) : Выражение))
    alias Выражение типВозврата;
  else
  static if (is(typeof(mixin(имяКласса)) : УзелТипа))
    alias УзелТипа типВозврата;
  else
    alias Узел типВозврата;
}

/// Generate functions which do the second отправь.
///
/// E.g.:
/// ---
/// Выражение visitCommaExpression(Визитёр визитёр, ВыражениеЗапятая c)
/// { визитёр.посети(c); /* Second отправь. */ }
/// ---
/// The equivalent in the traditional визитёр pattern would be:
/// ---
/// class ВыражениеЗапятая : Выражение
/// {
///   проц  accept(Визитёр визитёр)
///   { визитёр.посети(this); }
/// }
/// ---
ткст генерируйФункцииОтправки()
{
  ткст текст;
  foreach (имяКласса; г_именаКлассов)
    текст ~= "типВозврата!(\""~имяКласса~"\") посети"~имяКласса~"(Визитёр
визитёр, "~имяКласса~" c)\n"
             "{ return визитёр.посети(c); }\n";
  return текст;
}
// pragma(сооб, генерируйФункцииОтправки());

/++
 Generates an массив of function pointers.
```

```
 ---
 [
   cast(проц *)&visitCommaExpression,
   // etc.
 ]
 ---
+/
ткст генерируйВТаблицу()
{
  ткст текст = "[";
  foreach (имяКласса; г_именаКлассов)
    текст ~= "cast(ук)&посети"~имяКласса~",\n";
  return текст[0..$-2]~"]"; // срез away last ",\n"
}
// pragma(сооб, генерируйВТаблицу());

/// Implements a variation of the визитёр pattern.
///
/// Inherited by classes that need в traverse a D syntax tree
/// and do computations, transformations and другой things on it.
abstract class Визитёр
{
  mixin(генерируйМетодыВизита());

  static
    mixin(генерируйФункцииОтправки());

  // Это необходимо, поскольку компилятор помещает
  // данный массив в сегмент статических данных.
  mixin("private const _dispatch_vtable = " ~ генерируйВТаблицу() ~ ";");
  /// The таблица holding function pointers в the second отправь functions.
  static const отправь_втаблицу = _dispatch_vtable;
  static assert(отправь_втаблицу.length == г_именаКлассов.length,
                "длина втаблицы не соответствует числу классов");

  /// Looks up the second отправь function for n and returns that.
  Узел function(Визитёр, Узел) дайФункциюОтправки()(Узел n)
  {
    return cast(Узел function(Визитёр, Узел))отправь_втаблицу[n.вид];
  }

  /// The main and first отправь function.
  Узел отправь(Узел n)
  { // Second отправь is done in the called function.
    return дайФункциюОтправки(n)(this, n);
  }

final:
  Декларация посети(Декларация n)
  { return посетиД(n); }
  Инструкция посети(Инструкция n)
  { return посетиИ(n); }
  Выражение посети(Выражение n)
  { return посетиВ(n); }
  УзелТипа посети(УзелТипа n)
  { return посетиТ(n); }
  Узел посети(Узел n)
  { return посетиУ(n); }

  Декларация посетиД(Декларация n)
  {
    return cast(Декларация)cast(ук)отправь(n);
  }
```

```
  Инструкция посетиИ(Инструкция n)
  {
    return cast(Инструкция)cast(ук)отправь(n);
  }

  Выражение посетиВ(Выражение n)
  {
    return cast(Выражение)cast(ук)отправь(n);
  }

  УзелТипа посетиТ(УзелТипа n)
  {
    return cast(УзелТипа)cast(ук)отправь(n);
  }

  Узел посетиУ(Узел n)
  {
    return отправь(n);
  }
}
```

Пакет Семантика (semantic)

```
module drc.semantic.Analysis;

import drc.ast.Node,
       drc.ast.Expressions;
import drc.semantic.Scope;
import drc.lexer.IdTable;
import drc.Compilation;
import common;

/// Общая семантика для декларации прагм и инструкций.
проц  семантикаПрагмы(Масштаб масш, Сема* pragmaLoc,
                      Идентификатор* идент,
                      Выражение[] арги)
{
  if (идент is Идент.сооб)
    прагма_сооб(масш, pragmaLoc, арги);
  else if (идент is Идент.lib)
    прагма_биб(масш, pragmaLoc, арги);
  // else
  //   масш.ошибка(начало, "unrecognized pragma");
}

/// Оценивает прагму msg (сообщение).
проц  прагма_сооб(Масштаб масш, Сема* pragmaLoc, Выражение[] арги)
{
  if (арги.length == 0)
    return /*масш.ошибка(pragmaLoc, "ожидаемое выражение arguments в
pragma")*/;

  foreach (арг; арги)
  {
    auto в = арг/+.evaluate()+/;
    if (в is null)
    {
```

```
      // масш.ошибка(в.начало, "выражение is not оцениuatable at compile
время");
    }
    else if (auto ткстВыр = в.Является!(ТекстовоеВыражение))
      // Печать текста на стандартный вывод.
      выдай(ткстВыр.дайТекст());
    else
    {
      // масш.ошибка(в.начало, "выражение must evaluate в a ткст");
    }
  }
  // Print a нс at the конец.
  выдай('\n');
}

/// Оценивает прагму lib (биб).
проц  прагма_биб(Масштаб масш, Сема* pragmaLoc, Выражение[] арги)
{
  if (арги.length != 1)
    return /*масш.ошибка(pragmaLoc, "ожидаемое one выражение аргумент в
pragma")*/;

  auto в = арги[0]/+.evaluate()+/;
  if (в is null)
  {
    // масш.ошибка(в.начало, "выражение is not оцениuatable at compile
время");
  }
  else if (auto ткстВыр = в.Является!(ТекстовоеВыражение))
  {
    // TODO: collect library пути in Модуль?
    // масш.модуль.addLibrary(ткстВыр.дайТекст());
  }
  else
  {
    // масш.ошибка(в.начало, "выражение must evaluate в a ткст");
  }
}

/// Возвращает да, если должна компилироваться первая ветвь(отладочной
декларации/инструкции); или
/// нет, если нужно компилировать ветвь else.
бул выборОтладВетви(Сема* услов, КонтекстКомпиляции контекст)
{
  if (услов)
  {
    if (услов.вид == ТОК.Идентификатор)
    {
      if (контекст.найдиИдОтладки(услов.идент.ткт))
        return да;
    }
    else if (услов.бцел_ <= контекст.уровеньОтладки)
      return да;
  }
  else if (1 <= контекст.уровеньОтладки)
    return да;
  return нет;
}

/// Returns да if the first branch (of a version declaration/statement) or
/// нет if the else-branch should be compiled in.
бул выборВерсионВетви(Сема* услов, КонтекстКомпиляции контекст)
{
```

```
  assert(услов);
  if (услов.вид == ТОК.Идентификатор || услов.вид == ТОК.Юниттест)
  {
    if (контекст.найдиИдВерсии(услов.идент.ткт))
      return да;
  }
  else if (услов.бцел_ >= контекст.уровеньВерсии)
    return да;
  return нет;
}
```

---

module drc.semantic.Module;

```
import drc.ast.Node,
       drc.ast.Declarations;
import drc.parser.Parser;
import drc.lexer.Lexer,
       drc.lexer.IdTable;
import drc.semantic.Symbol,
       drc.semantic.Symbols;
import drc.Location;
import drc.Messages;
import drc.Diagnostics;
import drc.SourceText;
import common;

import io.FilePath;
import io.model;

alias ФайлКонст.СимПутьРазд папРазд;

/// Представляет модуль семантики и исходный файл.
class Модуль : СимволМасштаба
{
  ИсходныйТекст исходныйТекст; /// Исходный файл данного модуля.
  ткст пкиМодуля; /// Fully qualified имя of the module. E.g.: drc.ast.Node
  ткст имяПакета; /// E.g.: drc.ast
  ткст имяМодуля; /// E.g.: Узел

  СложнаяДекларация корень; /// The корень of the разбор tree.
  ДекларацияИмпорта[] импорты; /// ДекларацииИмпорта found in this file.
  ДекларацияМодуля деклМодуля; /// The optional ДекларацияМодуля in this
file.
  Парсер парсер; /// The парсер used в разбор this file.

  /// Indicates which passes have been пуск on this module.
  ///
  /// 0 = no pass$(BR)
  /// 1 = semantic pass 1$(BR)
  /// 2 = semantic pass 2
  бцел семантическийПроходка;
  Модуль[] модули; /// The imported модули.

  Диагностика диаг; /// Collects ошибка сообщения.

  this()
  {
    super(СИМ.Модуль, null, null);
  }
```

```
/// Строит Модуль объект.
/// Параметры:
///   путьКФайлу = file путь в the source текст; loaded in the constructor.
///   диаг = used for collecting ошибка сообщения.
this(ткст путьКФайлу, Диагностика диаг = null)
{
  this();
  this.исходныйТекст = new ИсходныйТекст(путьКФайлу);
  this.диаг = диаг is null ? new Диагностика() : диаг;
  this.исходныйТекст.загрузи(диаг);
}

/// Возвращает file путь of the source текст.
ткст путьКФайлу()
{
  return исходныйТекст.путьКФайлу;
}

/// Возвращает file extension: "d" or "di".
ткст расширениеФайла()
{
  foreach_reverse(i, c; путьКФайлу)
    if (c == '.')
      return путьКФайлу[i+1..$];
  return "";
}

/// Sets the парсер в be used for parsing the source текст.
проц  установиПарсер(Парсер парсер)
{
  this.парсер = парсер;
}

/// Parses the module.
/// Бросьs:
///   An Exception if the there's no ДекларацияМодуля and
///   the file имя is an invalid or reserved D identifier.
проц  разбор()
{
  if (this.парсер is null)
    this.парсер = new Парсер(исходныйТекст, диаг);

  this.корень = парсер.старт();
  this.импорты = парсер.импорты;

  // Set the fully qualified имя of this module.
  if (this.корень.отпрыски.length)
  { // деклМодуля will be null if first узел isn't a ДекларацияМодуля.
    this.деклМодуля = this.корень.отпрыски[0].Является!(ДекларацияМодуля);
    if (this.деклМодуля)
      this.установиПКН(деклМодуля.дайПКН()); // E.g.: drc.ast.Node
  }

  if (!this.пкиМодуля.length)
  { // Take the base имя of the file as the module имя.
    auto ткт = (new ФПуть(путьКФайлу)).имя(); // E.g.: Узел
    if (!Лексер.действитНерезИдентификатор_ли(ткт))
    {
      auto положение = this.перваяСема().дайПоложениеОшибки();
      auto сооб = Формат(сооб.НеверноеИмяМодуля, ткт);
      диаг ~= new ОшибкаЛексера(положение, сооб);
      ткт = ТаблицаИд.генИдМодуля().ткт;
    }
```

```
    this.пкиМодуля = this.имяМодуля = ткт;
  }
  assert(this.пкиМодуля.length);

  // Set the символ имя.
  this.имя = ТаблицаИд.сыщи(this.имяМодуля);
}

/// Возвращает first сема of the module's source текст.
Сема* перваяСема()
{
  return парсер.лексер.перваяСема();
}

/// Возвращает начало сема of the module declaration
/// or, if it doesn't exist, the first сема in the source текст.
Сема* дайСемуДеклМодуля()
{
  return деклМодуля ? деклМодуля.начало : перваяСема();
}

/// Returns да if there are ошибки in the source file.
бул естьОшибки()
{
  return парсер.ошибки.length || парсер.лексер.ошибки.length;
}

/// Returns a список of import пути.
/// E.g.: ["dil/ast/Узел", "dil/semantic/Модуль"]
ткст[] дайПутиИмпорта()
{
  ткст[] результат;
  foreach (import_; импорты)
    результат ~= import_.дайПКНМодуля(папРазд);
  return результат;
}

/// Возвращает fully qualified имя of this module.
/// E.g.: drc.ast.Node
ткст дайПКН()
{
  return пкиМодуля;
}

/// Set's the module's ПКИ.
проц  установиПКН(ткст пкиМодуля)
{
  бцел i = пкиМодуля.length;
  if (i != 0) // Don't decrement if ткст has zero length.
    i--;
  // Find last dot.
  for (; i != 0 && пкиМодуля[i] != '.'; i--)
  {}
  this.пкиМодуля = пкиМодуля;
  if (i == 0)
    this.имяМодуля = пкиМодуля; // No dot found.
  else
  {
    this.имяПакета = пкиМодуля[0..i];
    this.имяМодуля = пкиМодуля[i+1..$];
  }
}
```

```
  /// Возвращает module's ПКИ with slashes instead of dots.
  /// E.g.: dil/ast/Узел
  ткст дайПутьПКН()
  {
    ткст FQNPath = пкиМодуля.dup;
    foreach (i, c; FQNPath)
      if (c == '.')
        FQNPath[i] = папРазд;
    return FQNPath;
  }
}
```

```
module drc.semantic.Package;

import drc.semantic.Symbol,
       drc.semantic.Symbols,
       drc.semantic.Module;
import drc.lexer.IdTable;
import common;

/// Пакетная группа модулей и иные пакеты.
class Пакет : СимволМасштаба
{
  ткст имяПкт;    /// Название пакета. Напр.: 'dil'.
  Пакет[] пакеты; /// Подпакеты в данном пакете.
  Модуль[] модули;   /// Модули данного пакета.

  /// Строит Пакет объект.
  this(ткст имяПкт)
  {
    auto идент = ТаблицаИд.сыщи(имяПкт);
    super(СИМ.Пакет, идент, null);
    this.имяПкт = имяПкт;
  }

  /// Возвращает да, если пакет корневой.
  бул корень_ли()
  {
    return родитель is null;
  }

  /// Возвращает пакет-родитель или пусто, если это корневой пакет.
  Пакет пакетРодитель()
  {
    if (корень_ли())
      return null;
    assert(родитель.Пакет_ли);
    return родитель.в!(Пакет);
  }

  /// Добавляет модуль в данный пакет.
  проц  добавь(Модуль модуль)
  {
    модуль.родитель = this;
    модули ~= модуль;
    вставь(модуль, модуль.имя);
  }

  /// Добавляет пакет в данный пакет.
  проц  добавь(Пакет пкт)
  {
```

```
    пкт.родитель = this;
    пакеты ~= пкт;
    вставь(пкт, пкт.имя);
  }
}
```

module drc.semantic.Pass1;

```
import  drc.ast.Visitor,
        drc.ast.Node,
        drc.ast.Declarations,
        drc.ast.Expressions,
        drc.ast.Statements,
        drc.ast.Types,
        drc.ast.Parameters;
import drc.lexer.IdTable;
import drc.semantic.Symbol,
        drc.semantic.Symbols,
        drc.semantic.Types,
        drc.semantic.Scope,
        drc.semantic.Module,
        drc.semantic.Analysis;
import drc.Compilation;
import drc.Diagnostics;
import drc.Messages;
import drc.Enums;
import drc.CompilerInfo;
import common;

import io.model;
alias ФайлКонст.СимПутьРазд папРазд;

/// Первая проходка - проходка по декларациям.
///
/// Основная задача класса - проход по дереву разбора,
/// нахождение всех видов деклараций и добавление их
/// в таблицы символов соответствующих им масштабов.
class СемантическаяПроходка1 : Визитёр
{
  Масштаб масш; /// Текущий Масштаб.
  Модуль модуль; /// Модуль, подлежащий семантической проверке.
  КонтекстКомпиляции контекст; /// Контекст компиляции.
  Модуль delegate(ткст) импортируйМодуль; /// Вызывается при импорте модуля.

  // Attributes:
  ТипКомпоновки типКомпоновки; /// Текущий тип компоновки.
  Защита защита; /// Текущий атрибут защиты.
  КлассХранения классХранения; /// Текущие классы хранения.
  бцел размерРаскладки; /// Текущий align размер.

  /// Строит СемантическаяПроходка1 объект.
  /// Параметры:
  ///   модуль = обрабатываемый модуль.
  ///   контекст = контекст компиляции.
  this(Модуль модуль, КонтекстКомпиляции контекст)
  {
    this.модуль = модуль;
    this.контекст = new КонтекстКомпиляции(контекст);
    this.размерРаскладки = контекст.раскладкаСтруктуры;
  }

  /// Начинает обработку модуля.
  проц  пуск()
```

```
{
  assert(модуль.корень !is null);
  // Create module Масштаб.
  масш = new Масштаб(null, модуль);
  модуль.семантическийПроходка = 1;
  посети(модуль.корень);
}

/// Входит в новый Масштаб.
проц  войдиВМасштаб(СимволМасштаба s)
{
  масш = масш.войдиВ(s);
}

/// Выходит из текущего Масштаба.
проц  выйдиИзМасштаба()
{
  масш = масш.выход();
}

/// Возвращает да, если символ на уровне модульного масштаба.
бул масштабМодуля_ли()
{
  return масш.символ.Модуль_ли();
}

/// Вставляет символ в текущий Масштаб.
проц  вставь(Символ символ)
{
  вставь(символ, символ.имя);
}

/// Вставляет символ в текущий Масштаб.
проц  вставь(Символ символ, Идентификатор* имя)
{
  auto symX = масш.символ.сыщи(имя);
  if (symX)
    сообщиОКонфликтеСимволов(символ, symX, имя);
  else
    масш.символ.вставь(символ, имя);
  // Set the current Масштаб символ as the родитель.
  символ.родитель = масш.символ;
}

/// Вставляет символ в симМасшт.
проц  вставь(Символ символ, СимволМасштаба симМасшт)
{
  auto symX = симМасшт.сыщи(символ.имя);
  if (symX)
    сообщиОКонфликтеСимволов(символ, symX, символ.имя);
  else
    симМасшт.вставь(символ, символ.имя);
  // Set the current Масштаб символ as the родитель.
  символ.родитель = симМасшт;
}

/// Вставляет символ в текущий Масштаб с перегрузкой имени.
проц  вставьПерегрузку(Символ сим)
{
  auto имя = сим.имя;
  auto сим2 = масш.символ.сыщи(имя);
  if (сим2)
  {
```

```
      if (сим2.НаборПерегрузки_ли)
        (cast(НаборПерегрузки)cast(ук)сим2).добавь(сим);
      else
        сообщиОКонфликтеСимволов(сим, сим2, имя);
    }
    else
      // Create a new overload установи.
      масш.символ.вставь(new НаборПерегрузки(имя, сим.узел), имя);
    // Set the current Масштаб символ as the родитель.
    сим.родитель = масш.символ;
  }

  /// Создаёт отчёт об ошибке: новый символ s1 конфликтует с существующим
символом s2.
  проц  сообщиОКонфликтеСимволов(Символ s1, Символ s2, Идентификатор* имя)
  {
    auto место = s2.узел.начало.дайПоложениеОшибки();
    auto locString = Формат("{}({},{})", место.путьКФайлу, место.номСтр,
место.номСтолб);
    ошибка(s1.узел.начало, сооб.ДеклКонфликтуетСДекл, имя.ткт, locString);
  }

  /// Создаёт отчёт об ошибке.
  проц  ошибка(Сема* сема, ткст форматирСооб, ...)
  {
    if (!модуль.диаг)
      return;
    auto положение = сема.дайПоложениеОшибки();
    auto сооб = Формат(_arguments, _argptr, форматирСооб);
    модуль.диаг ~= new ОшибкаСемантики(положение, сооб);
  }


  /// Собирает инфу об узлах, оценка которых будет проведена позже.
  static class Иной
  {
    Узел узел;
    СимволМасштаба символ;
    // Saved attributes.
    ТипКомпоновки типКомпоновки;
    Защита защита;
    КлассХранения классХранения;
    бцел размерРаскладки;
  }

  /// Список объявлений mixin, static if, static assert и pragma(сооб,...).
  ///
  /// Их анализ разделен, так как они следуют за
  /// оценкой  выражений.
  Иной[] deferred;

  /// Добавляет deferred узел в the список.
  проц  добавьИной(Узел узел)
  {
    auto d = new Иной;
    d.узел = узел;
    d.символ = масш.символ;
    d.типКомпоновки = типКомпоновки;
    d.защита = защита;
    d.классХранения = классХранения;
    d.размерРаскладки = размерРаскладки;
    deferred ~= d;
  }
```

```
  private alias Декларация Д; /// A handy alias. Saves typing.

override
{
  Д посети(СложнаяДекларация d)
  {
    foreach (декл; d.деклы)
      посетиД(декл);
    return d;
  }

  Д посети(НелегальнаяДекларация)
  { assert(0, "semantic pass on invalid AST"); return null; }

  // Д посети(ПустаяДекларация ed)
  // { return ed; }

  // Д посети(ДекларацияМодуля)
  // { return null; }

  Д посети(ДекларацияИмпорта d)
  {
    if (импортируйМодуль is null)
      return d;
    foreach (путьПоПКНМодуля; d.дайПКНМодуля(папРазд))
    {
      auto importedModule = импортируйМодуль(путьПоПКНМодуля);
      if (importedModule is null)
        ошибка(d.начало, сооб.МодульНеЗагружен, путьПоПКНМодуля ~ ".d");
      модуль.модули ~= importedModule;
    }
    return d;
  }

  Д посети(ДекларацияАлиаса ad)
  {
    return ad;
  }

  Д посети(ДекларацияТипдефа td)
  {
    return td;
  }

  Д посети(ДекларацияПеречня d)
  {
    if (d.символ)
      return d;

    // Create the символ.
    d.символ = new Перечень(d.имя, d);

    бул анонимен_ли = d.символ.анонимен_ли;
    if (анонимен_ли)
      d.символ.имя = ТаблицаИд.генИДАнонПеречня();

    вставь(d.символ);

    auto parentScopeSymbol = масш.символ;
    auto enumSymbol = d.символ;
    войдиВМасштаб(d.символ);
    // Declare члены.
```

```
    foreach (член; d.члены)
    {
      посетиД(член);

      if (анонимен_ли) // Also вставь into родитель Масштаб if enum is
anonymous.
        вставь(член.символ, parentScopeSymbol);

      член.символ.тип = enumSymbol.тип; // Присвоить ТипПеречень.
    }
    выйдиИзМасштаба();
    return d;
  }

  Д посети(ДекларацияЧленаПеречня d)
  {
    d.символ = new ЧленПеречня(d.имя, защита, классХранения, типКомпоновки,
d);
    вставь(d.символ);
    return d;
  }

  Д посети(ДекларацияКласса d)
  {
    if (d.символ)
      return d;
    // Create the символ.
    d.символ = new Класс(d.имя, d);
    // Insert into current Масштаб.
    вставь(d.символ);
    войдиВМасштаб(d.символ);
    // Далее semantic analysis.
    d.деклы && посетиД(d.деклы);
    выйдиИзМасштаба();
    return d;
  }

  Д посети(ДекларацияИнтерфейса d)
  {
    if (d.символ)
      return d;
    // Create the символ.
    d.символ = new drc.semantic.Symbols.Интерфейс(d.имя, d);
    // Insert into current Масштаб.
    вставь(d.символ);
    войдиВМасштаб(d.символ);
      // Далее semantic analysis.
      d.деклы && посетиД(d.деклы);
    выйдиИзМасштаба();
    return d;
  }

  Д посети(ДекларацияСтруктуры d)
  {
    if (d.символ)
      return d;
    // Create the символ.
    d.символ = new Структура(d.имя, d);

    if (d.символ.анонимен_ли)
      d.символ.имя = ТаблицаИд.genAnonStructID();
    // Insert into current Масштаб.
    вставь(d.символ);
```

```
  войдиВМасштаб(d.символ);
    // Далее semantic analysis.
    d.деклы && посетиД(d.деклы);
  выйдиИзМасштаба();

  if (d.символ.анонимен_ли)
    // Insert члены into родитель Масштаб as well.
    foreach (член; d.символ.члены)
      вставь(член);
  return d;
}

Д посети(ДекларацияСоюза d)
{
  if (d.символ)
    return d;
  // Create the символ.
  d.символ = new Союз(d.имя, d);

  if (d.символ.анонимен_ли)
    d.символ.имя = ТаблицаИд.genAnonUnionID();

  // Insert into current Масштаб.
  вставь(d.символ);

  войдиВМасштаб(d.символ);
    // Далее semantic analysis.
    d.деклы && посетиД(d.деклы);
  выйдиИзМасштаба();

  if (d.символ.анонимен_ли)
    // Insert члены into родитель Масштаб as well.
    foreach (член; d.символ.члены)
      вставь(член);
  return d;
}

Д посети(ДекларацияКонструктора d)
{
  auto func = new Функция(Идент.Ктор, d);
  вставьПерегрузку(func);
  return d;
}

Д посети(ДекларацияСтатическогоКонструктора d)
{
  auto func = new Функция(Идент.Ктор, d);
  вставьПерегрузку(func);
  return d;
}

Д посети(ДекларацияДеструктора d)
{
  auto func = new Функция(Идент.Дтор, d);
  вставьПерегрузку(func);
  return d;
}

Д посети(ДекларацияСтатическогоДеструктора d)
{
  auto func = new Функция(Идент.Дтор, d);
  вставьПерегрузку(func);
```

```
    return d;
  }

  Д посети(ДекларацияФункции d)
  {
    auto func = new Функция(d.имя, d);
    вставьПерегрузку(func);
    return d;
  }

  Д посети(ДекларацияПеременных vd)
  {
    // Ошибка if we are in an interface.
    if (масш.символ.Интерфейс_ли && !vd.статический_ли)
      return ошибка(vd.начало, сооб.УИнтерфейсаНеДолжноБытьПеременных), vd;

    // Insert переменная символы in this declaration into the символ таблица.
    foreach (i, имя; vd.имена)
    {
      auto переменная = new Переменная(имя, защита, классХранения,
типКомпоновки, vd);
      переменная.значение = vd.инициализаторы[i];
      vd.переменные ~= переменная;
      вставь(переменная);
    }
    return vd;
  }

  Д посети(ДекларацияИнварианта d)
  {
    auto func = new Функция(Идент.Инвариант, d);
    вставь(func);
    return d;
  }

  Д посети(ДекларацияЮниттеста d)
  {
    auto func = new Функция(Идент.Юниттест, d);
    вставьПерегрузку(func);
    return d;
  }

  Д посети(ДекларацияОтладки d)
  {
    if (d.определение_ли)
    { // debug = Id | Цел
      if (!масштабМодуля_ли())
        ошибка(d.начало, сооб.DebugSpecModuleLevel, d.спец.исхТекст);
      else if (d.спец.вид == ТОК.Идентификатор)
        контекст.добавьИдОтладки(d.спец.идент.ткт);
      else
        контекст.уровеньОтладки = d.спец.бцел_;
    }
    else
    { // debug ( Condition )
      if (выборОтладВетви(d.услов, контекст))
        d.компилированныеДеклы = d.деклы;
      else
        d.компилированныеДеклы = d.деклыИначе;
      d.компилированныеДеклы && посетиД(d.компилированныеДеклы);
    }
    return d;
  }
```

```
Д посети(ДекларацияВерсии d)
{
  if (d.определение_ли)
  { // version = Id | Цел
    if (!масштабМодуля_ли())
      ошибка(d.начало, сооб.VersionSpecModuleLevel, d.спец.исхТекст);
    else if (d.спец.вид == ТОК.Идентификатор)
      контекст.добавьИдВерсии(d.спец.идент.ткт);
    else
      контекст.уровеньВерсии = d.спец.бцел_;
  }
  else
  { // version ( Condition )
    if (выборВерсионВетви(d.услов, контекст))
      d.компилированныеДеклы = d.деклы;
    else
      d.компилированныеДеклы = d.деклыИначе;
    d.компилированныеДеклы && посетиД(d.компилированныеДеклы);
  }
  return d;
}

Д посети(ДекларацияШаблона d)
{
  if (d.символ)
    return d;
  // Create the символ.
  d.символ = new Шаблон(d.имя, d);
  // Insert into current Масштаб.
  вставьПерегрузку(d.символ);
  return d;
}

Д посети(ДекларацияНов d)
{
  auto func = new Функция(Идент.Нов, d);
  вставь(func);
  return d;
}

Д посети(ДекларацияУдали d)
{
  auto func = new Функция(Идент.Удалить, d);
  вставь(func);
  return d;
}

// Attributes:

Д посети(ДекларацияЗащиты d)
{
  auto saved = защита; // Save.
  защита = d.защ; // Set.
  посетиД(d.деклы);
  защита = saved; // Restore.
  return d;
}

Д посети(ДекларацияКлассаХранения d)
{
  auto saved = классХранения; // Save.
  классХранения = d.классХранения; // Set.
```

```
    посетиД(d.деклы);
    классХранения = saved; // Restore.
    return d;
  }

  Д посети(ДекларацияКомпоновки d)
  {
    auto saved = типКомпоновки; // Save.
    типКомпоновки = d.типКомпоновки; // Set.
    посетиД(d.деклы);
    типКомпоновки = saved; // Restore.
    return d;
  }

  Д посети(ДекларацияРазложи d)
  {
    auto saved = размерРаскладки; // Save.
    размерРаскладки = d.размер; // Set.
    посетиД(d.деклы);
    размерРаскладки = saved; // Restore.
    return d;
  }

  // Иной declarations:

  Д посети(ДекларацияСтатическогоПодтверди d)
  {
    добавьИной(d);
    return d;
  }

  Д посети(ДекларацияСтатическогоЕсли d)
  {
    добавьИной(d);
    return d;
  }

  Д посети(ДекларацияСмеси d)
  {
    добавьИной(d);
    return d;
  }

  Д посети(ДекларацияПрагмы d)
  {
    if (d.идент is Идент.сооб)
      добавьИной(d);
    else
    {
      семантикаПрагмы(масш, d.начало, d.идент, d.арги);
      посетиД(d.деклы);
    }
    return d;
  }
} // override
}
```

---

module drc.semantic.Pass2;

```
import drc.ast.DefaultVisitor,
       drc.ast.Node,
       drc.ast.Declarations,
```

```
       drc.ast.Expressions,
       drc.ast.Statements,
       drc.ast.Types,
       drc.ast.Parameters;
import drc.lexer.Identifier;
import drc.semantic.Symbol,
       drc.semantic.Symbols,
       drc.semantic.Types,
       drc.semantic.Scope,
       drc.semantic.Module,
       drc.semantic.Analysis;
import drc.code.Interpreter;
import drc.parser.Parser;
import drc.SourceText;
import drc.Diagnostics;
import drc.Messages;
import drc.Enums;
import drc.CompilerInfo;
import common;

/// Вторая проходка определяет типы символы и типы
/// выражений, а также оценивает их.
class СемантическаяПроходка2 : ДефолтныйВизитёр
{
  Масштаб масш; /// Текущий Масштаб.
  Модуль модуль; /// Модуль, подлежащий семантической проверке.

  /// Строит СемантическаяПроходка2 объект.
  /// Параметры:
  ///   модуль = проверяемый модуль.
  this(Модуль модуль)
  {
    this.модуль = модуль;
  }

  /// Начало семантического анализа.
  проц  пуск()
  {
    assert(модуль.корень !is null);
    // Create module Масштаб.
    масш = new Масштаб(null, модуль);
    модуль.семантическийПроходка = 2;
    посети(модуль.корень);
  }

  /// Вход в новый Масштаб.
  проц  войдиВМасштаб(СимволМасштаба s)
  {
    масш = масш.войдиВ(s);
  }

  /// Выход из текущего Масштаба.
  проц  выйдиИзМасштаба()
  {
    масш = масш.выход();
  }

  /// Оценивает и возвращает результат.
  Выражение интерпретируй(Выражение в)
  {
    return Интерпретатор.интерпретируй(в, модуль.диаг);
  }
```

```
  /// Создание отчёта об ошибке.
  проц  ошибка(Сема* сема, ткст форматирСооб, ...)
  {
    auto положение = сема.дайПоложениеОшибки();
    auto сооб = Формат(_arguments, _argptr, форматирСооб);
    модуль.диаг ~= new ОшибкаСемантики(положение, сооб);
  }

  /// Some handy aliases.
  private alias Декларация D;
  private alias Выражение E; /// определено
  private alias Инструкция S; /// определено
  private alias УзелТипа T; /// определено

  /// The current Масштаб символ в use for looking up identifiers.
  /// E.g.:
  /// ---
  /// объект.method(); // *) объект is looked up in the current Масштаб.
  ///                  // *) идМасштаб is установи if объект is a
СимволМасштаба.
  ///                  // *) method will be looked up in идМасштаб.
  /// drc.ast.Node.Узел узел; // A fully qualified тип.
  /// ---
  СимволМасштаба идМасштаб;

  /// Searches for a символ.
  Символ ищи(Сема* идСем)
  {
    assert(идСем.вид == ТОК.Идентификатор);
    auto ид = идСем.идент;
    Символ символ;

    if (идМасштаб is null)
      символ = масш.ищи(ид);
    else
      символ = идМасштаб.сыщи(ид);

    if (символ is null)
      ошибка(идСем, сооб.НеопределенныйИдентификатор, ид.ткт);
    else if (auto масшСимвол = cast(СимволМасштаба)символ)
      идМасштаб = масшСимвол;

    return символ;
  }

override
{
  D посети(СложнаяДекларация d)
  {
    return super.посети(d);
  }

  D посети(ДекларацияПеречня d)
  {
    d.символ.устОбрабатывается();

    Тип тип = Типы.Цел; // Дефолт в цел.
    if (d.типОснова)
      тип = посетиТ(d.типОснова).тип;
    // Set the enum's base тип.
    d.символ.тип.типОснова = тип;

    // TODO: check base тип. must be basic тип or another enum.
```

```
    войдиВМасштаб(d.символ);

    foreach (член; d.члены)
    {
      Выражение финальнЗначение;
      член.символ.устОбрабатывается();
      if (член.значение)
      {
        член.значение = посетиВ(член.значение);
        финальнЗначение = интерпретируй(член.значение);
        if (финальнЗначение is Интерпретатор.НЕИ)
          финальнЗначение = new ЦелВыражение(0, d.символ.тип);
      }
      //else
        // TODO: инкремент а число переменная and assign that в значение.
      член.символ.значение = финальнЗначение;
      член.символ.устОбработан();
    }

    выйдиИзМасштаба();
    d.символ.устОбработан();
    return d;
  }

  D посети(ДекларацияСмеси md)
  {
    if (md.деклы)
      return md.деклы;
    if (md.выражениеСмеси_ли)
    {
      md.аргумент = посетиВ(md.аргумент);
      auto выр = интерпретируй(md.аргумент);
      if (выр is Интерпретатор.НЕИ)
        return md;
      auto ткстВыр = выр.Является!(ТекстовоеВыражение);
      if (ткстВыр is null)
      {
        ошибка(md.начало, сооб.АргументСмесиДБТекстом);
        return md;
      }
      else
      { // Parse the declarations in the ткст.
        auto место = md.начало.дайПоложениеОшибки();
        auto путьКФайлу = место.путьКФайлу;
        auto исходныйТекст = new ИсходныйТекст(путьКФайлу,
ткстВыр.дайТекст());
        auto парсер = new Парсер(исходныйТекст, модуль.диаг);
        md.деклы = парсер.старт();
      }
    }
    else
    {
      // TODO: implement template mixin.
    }
    return md.деклы;
  }

  // Тип nodes:

  T посети(ТТип t)
  {
    t.в = посетиВ(t.в);
```

```
    t.тип = t.в.тип;
    return t;
  }

  Т посети(ТМассив t)
  {
    auto типОснова = посетиТ(t.следщ).тип;
    if (t.ассоциативный_ли)
      t.тип = типОснова.массивИз(посетиТ(t.ассоцТип).тип);
    else if (t.динамический_ли)
      t.тип = типОснова.массивИз();
    else if (t.статический_ли)
    {}
    else
      assert(t.срез_ли);
    return t;
  }

  Т посети(ТУказатель t)
  {
    t.тип = посетиТ(t.следщ).тип.укНа();
    return t;
  }

  Т посети(КвалифицированныйТип t)
  {
    if (t.лв.Является!(КвалифицированныйТип) is null)
      идМасштаб = null; // Reset at левый-most тип.
    посетиТ(t.лв);
    посетиТ(t.пв);
    t.тип = t.пв.тип;
    return t;
  }

  Т посети(ТИдентификатор t)
  {
    auto идСема = t.начало;
    auto символ = ищи(идСема);
    // TODO: save символ or its тип in t.
    return t;
  }

  Т посети(ТЭкземплярШаблона t)
  {
    auto идСема = t.начало;
    auto символ = ищи(идСема);
    // TODO: save символ or its тип in t.
    return t;
  }

  Т посети(ТМасштабМодуля t)
  {
    идМасштаб = модуль;
    return t;
  }

  Т посети(ИнтегральныйТип t)
  {
    // A таблица mapping the вид of a сема в its corresponding semantic Тип.
    ТипБазовый[ТОК] семВТип = [
      ТОК.Сим : Типы.Сим,   ТОК.Шим : Типы.Шим,   ТОК.Дим : Типы.Дим, ТОК.Бул
: Типы.Бул,
```

```
      ТОК.Байт : Типы.Байт,   ТОК.Ббайт : Типы.Ббайт,   ТОК.Крат : Типы.Крат,
ТОК.Бкрат : Типы.Бкрат,
      ТОК.Цел : Типы.Цел,    ТОК.Бцел : Типы.Бцел,    ТОК.Дол : Типы.Дол,
ТОК.Бдол : Типы.Бдол,
      ТОК.Цент : Типы.Цент,   ТОК.Бцент : Типы.Бцент,
      ТОК.Плав : Типы.Плав,  ТОК.Дво : Типы.Дво,  ТОК.Реал : Типы.Реал,
      ТОК.Вплав : Типы.Вплав, ТОК.Вдво : Типы.Вдво, ТОК.Вреал : Типы.Вреал,
      ТОК.Кплав : Типы.Кплав, ТОК.Кдво : Типы.Кдво, ТОК.Креал : Типы.Креал,
ТОК.Проц : Типы.Проц
    ];
    assert(т.лекс in семВТип);
    т.тип = семВТип[т.лекс];
    return т;
  }

  // Выражение nodes:

  Е посети(ВыражениеРодит в)
  {
    if (!в.тип)
    {
      в.следщ = посетиВ(в.следщ);
      в.тип = в.следщ.тип;
    }
    return в;
  }

  Е посети(ВыражениеЗапятая в)
  {
    if (!в.тип)
    {
      в.лв = посетиВ(в.лв);
      в.пв = посетиВ(в.пв);
      в.тип = в.пв.тип;
    }
    return в;
  }

  Е посети(ВыражениеИлиИли)
  { return null; }

  Е посети(ВыражениеИИ)
  { return null; }

  Е посети(ВыражениеСпецСема в)
  {
    if (в.тип)
      return в.значение;
    switch (в.особаяСема.вид)
    {
    case ТОК.СТРОКА, ТОК.ВЕРСИЯ:
      в.значение = new ЦелВыражение(в.особаяСема.бцел_, Типы.Бцел);
      break;
    case ТОК.ФАЙЛ, ТОК.ДАТА, ТОК.ВРЕМЯ, ТОК.ШТАМПВРЕМЕНИ, ТОК.ПОСТАВЩИК:
      в.значение = new ТекстовоеВыражение(в.особаяСема.ткт);
      break;
    default:
      assert(0);
    }
    в.тип = в.значение.тип;
    return в.значение;
  }
```

```
E посети(ВыражениеДоллар в)
{
  if (в.тип)
    return в;
  в.тип = Типы.Т_мера;
  // if (!inArraySubscript)
  //   ошибка("$ can only be in an массив subscript.");
  return в;
}

E посети(ВыражениеНуль в)
{
  if (!в.тип)
    в.тип = Типы.Проц_ук;
  return в;
}

E посети(БулевоВыражение в)
{
  if (в.тип)
    return в;
  в.значение = new ЦелВыражение(в.вБул(), Типы.Бул);
  в.тип = Типы.Бул;
  return в;
}

E посети(ЦелВыражение в)
{
  if (в.тип)
    return в;

  if (в.число & 0x8000_0000_0000_0000)
    в.тип = Типы.Бдол; // 0xFFFF_FFFF_FFFF_FFFF
  else if (в.число & 0xFFFF_FFFF_0000_0000)
    в.тип = Типы.Дол; // 0x7FFF_FFFF_FFFF_FFFF
  else if (в.число & 0x8000_0000)
    в.тип = Типы.Бцел; // 0xFFFF_FFFF
  else
    в.тип = Типы.Цел; // 0x7FFF_FFFF
  return в;
}

E посети(ВыражениеРеал в)
{
  if (!в.тип)
    в.тип = Типы.Дво;
  return в;
}

E посети(ВыражениеКомплекс в)
{
  if (!в.тип)
    в.тип = Типы.Кдво;
  return в;
}

E посети(ВыражениеСим в)
{
  return в;
}

E посети(ТекстовоеВыражение в)
{
```

```
    return в;
  }

  E посети(ВыражениеСмесь me)
  {
    if (me.тип)
      return me.выр;
    me.выр = посетиВ(me.выр);
    auto выр = интерпретируй(me.выр);
    if (выр is Интерпретатор.НЕИ)
      return me;
    auto ткстВыр = выр.Является!(ТекстовоеВыражение);
    if (ткстВыр is null)
     ошибка(me.начало, сооб.АргументСмесиДБТекстом);
    else
    {
      auto место = me.начало.дайПоложениеОшибки();
      auto путьКФайлу = место.путьКФайлу;
      auto исходныйТекст = new ИсходныйТекст(путьКФайлу, ткстВыр.дайТекст());
      auto парсер = new Парсер(исходныйТекст, модуль.диаг);
      выр = парсер.старт2();
      выр = посетиВ(выр); // Check выражение.
    }
    me.выр = выр;
    me.тип = выр.тип;
    return me.выр;
  }

  E посети(ВыражениеИмпорта ie)
  {
    if (ie.тип)
      return ie.выр;
    ie.выр = посетиВ(ie.выр);
    auto выр = интерпретируй(ie.выр);
    if (выр is Интерпретатор.НЕИ)
      return ie;
    auto ткстВыр = выр.Является!(ТекстовоеВыражение);
    //if (ткстВыр is null)
    //  ошибка(me.начало, сооб.ImportArgumentMustBeString);
    // TODO: загрузи file
    //ie.выр = new ТекстовоеВыражение(loadImportFile(ткстВыр.дайТекст()));
    return ie.выр;
  }
}
}
```

```
/// Описание: Этот модуль присутствует в целях тестирования
/// иного алгоритма проведения семантического анализа,
/// для сравнения с СемантическаяПроходка1 и СемантическаяПроходка2!
module drc.semantic.Passes;

import drc.ast.DefaultVisitor,
       drc.ast.Node,
       drc.ast.Declarations,
       drc.ast.Expressions,
       drc.ast.Statements,
       drc.ast.Types,
       drc.ast.Parameters;
import drc.lexer.IdTable;
import drc.parser.Parser;
import drc.semantic.Symbol,
```

```
        drc.semantic.Symbols,
        drc.semantic.Types,
        drc.semantic.Scope,
        drc.semantic.Module,
        drc.semantic.Analysis;
import drc.code.Interpreter;
import drc.Compilation;
import drc.SourceText;
import drc.Diagnostics;
import drc.Messages;
import drc.Enums;
import drc.CompilerInfo;
import common;

/// Некоторые полезные замещения.
private alias Декларация Д;
private alias Выражение В; /// определено
private alias Инструкция И; /// определено
private alias УзелТипа Т; /// определено
private alias Параметр П; /// определено
private alias Узел У; /// определено

/// Базовый класс для иного класса семантических проходок.
abstract class СемантическаяПроходка : ДефолтныйВизитёр
{
  Масштаб масш; /// Текущий Масштаб.
  Модуль модуль; /// Семантически проверяемый модуль.
  КонтекстКомпиляции контекст; /// Контекст компиляции.

  /// Строит СемантическаяПроходка объект.
  /// Параметры:
  ///   модуль = обрабатываемый модуль.
  ///   контекст = контекст компиляции.
  this(Модуль модуль, КонтекстКомпиляции контекст)
  {
    this.модуль = модуль;
    this.контекст = контекст;
  }

  проц  пуск()
  {

  }

  /// Входит в новый Масштаб.
  проц  войдиВМасштаб(СимволМасштаба s)
  {
    масш = масш.войдиВ(s);
  }

  /// Выходит из текущего масштаба Масштаб.
  проц  выйдиИзМасштаба()
  {
    масш = масш.выход();
  }

  /// Возвращает да, если это модульный Масштаб.
  бул масштабМодуля_ли()
  {
    return масш.символ.Модуль_ли();
  }

  /// Вставляет символ в текущий Масштаб.
```

```
  проц  вставь(Символ символ)
  {
    вставь(символ, символ.имя);
  }

  /// Вставляет символ в текущий Масштаб.
  проц  вставь(Символ символ, Идентификатор* имя)
  {
    auto symX = масш.символ.сыщи(имя);
    if (symX)
      сообщиОКонфликтеСимволов(символ, symX, имя);
    else
      масш.символ.вставь(символ, имя);
    // Set the current Масштаб символ as the родитель.
    символ.родитель = масш.символ;
  }

  /// Вставляет символ в симМасшт.
  проц  вставь(Символ символ, СимволМасштаба симМасшт)
  {
    auto symX = симМасшт.сыщи(символ.имя);
    if (symX)
      сообщиОКонфликтеСимволов(символ, symX, символ.имя);
    else
      симМасшт.вставь(символ, символ.имя);
    // Set the current Масштаб символ as the родитель.
    символ.родитель = симМасшт;
  }

  /// Вставляет символ с перегрузкой имени в текущий Масштаб.
  проц  вставьПерегрузку(Символ сим)
  {
    auto имя = сим.имя;
    auto сим2 = масш.символ.сыщи(имя);
    if (сим2)
    {
      if (сим2.НаборПерегрузки_ли)
        (cast(НаборПерегрузки)cast(ук)сим2).добавь(сим);
      else
        сообщиОКонфликтеСимволов(сим, сим2, имя);
    }
    else
      // Create a new overload установи.
      масш.символ.вставь(new НаборПерегрузки(имя, сим.узел), имя);
    // Set the current Масштаб символ as the родитель.
    сим.родитель = масш.символ;
  }

  /// Репортирует об ошибке: новый символ s1 конфликтует с существующим
символом s2.
  проц  сообщиОКонфликтеСимволов(Символ s1, Символ s2, Идентификатор* имя)
  {
    auto место = s2.узел.начало.дайПоложениеОшибки();
    auto locString = Формат("{}({},{})", место.путьКФайлу, место.номСтр,
место.номСтолб);
    ошибка(s1.узел.начало, сооб.ДеклКонфликтуетСДекл, имя.ткт, locString);
  }

  /// Ошибка сообщения are reported for undefined identifiers if да.
  бул reportUndefinedIds;

  /// Incremented when an undefined identifier was found.
  бцел undefinedIdsCount;
```

```
  /// The символ that must be ignored an пропустиred during a символ ищи.
  Символ ignoreSymbol;

  /// The current Масштаб символ в use for looking up identifiers.
  /// B.g.:
  /// ---
  /// объект.method(); // *) объект is looked up in the current Масштаб.
  ///                  // *) идМасштаб is установи if объект is a
СимволМасштаба.
  ///                  // *) method will be looked up in идМасштаб.
  /// drc.ast.Node.Узел узел; // A fully qualified тип.
  /// ---
  СимволМасштаба идМасштаб;

  /// Этот объект is assigned в идМасштаб when a символ сыщи
  /// returned no valid символ.
  static const СимволМасштаба emptyIdScope;
  static this()
  {
    this.emptyIdScope = new СимволМасштаба();
  }

  // Sets a new идМасштаб символ.
  проц  setIdScope(Символ символ)
  {
    if (символ)
      if (auto масшСимвол = cast(СимволМасштаба)символ)
        return идМасштаб = масшСимвол;
    идМасштаб = emptyIdScope;
  }

  /// Searches for a символ.
  Символ ищи(Сема* идСем)
  {
    assert(идСем.вид == ТОК.Идентификатор);
    auto ид = идСем.идент;
    Символ символ;

    if (идМасштаб is null)
      // Search in the таблица of another символ.
      символ = ignoreSymbol ?
               масш.ищи(ид, ignoreSymbol) :
               масш.ищи(ид);
    else
      символ = идМасштаб.сыщи(ид);

    if (символ)
      return символ;

    if (reportUndefinedIds)
      ошибка(идСем, сооб.НеопределенныйИдентификатор, ид.ткт);
    undefinedIdsCount++;
    return null;
  }

  /// Creates an ошибка report.
  проц  ошибка(Сема* сема, ткст форматирСооб, ...)
  {
    if (!модуль.диаг)
      return;
    auto положение = сема.дайПоложениеОшибки();
    auto сооб = Формат(_arguments, _argptr, форматирСооб);
```

```
    модуль.диаг ~= new ОшибкаСемантики(положение, сооб);
  }
}

class ПерваяСемантическаяПроходка : СемантическаяПроходка
{
  Модуль delegate(ткст) импортируйМодуль; /// Called when importing a module.

  // Attributes:
  ТипКомпоновки типКомпоновки; /// Current linkage тип.
  Защита защита; /// Current защита attribute.
  КлассХранения классХранения; /// Current storage classes.
  бцел размерРаскладки; /// Current align размер.

  /// Строит СемантическаяПроходка объект.
  /// Параметры:
  ///   модуль = the module в be processed.
  ///   контекст = the compilation контекст.
  this(Модуль модуль, КонтекстКомпиляции контекст)
  {
    super(модуль, new КонтекстКомпиляции(контекст));
    this.размерРаскладки = контекст.раскладкаСтруктуры;
  }

  override проц  пуск()
  {
    assert(модуль.корень !is null);
    // Create module Масштаб.
    масш = new Масштаб(null, модуль);
    модуль.семантическийПроходка = 1;
    посетиУ(модуль.корень);
  }


/+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  |                                  Declarations
|

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+/

override
{
  Д посети(СложнаяДекларация d)
  {
    foreach (декл; d.деклы)
      посетиД(декл);
    return d;
  }

  Д посети(НелегальнаяДекларация)
  { assert(0, "semantic pass on invalid AST"); return null; }

  // Д посети(ПустаяДекларация ed)
  // { return ed; }

  // Д посети(ДекларацияМодуля)
  // { return null; }

  Д посети(ДекларацияИмпорта d)
  {
    if (импортируйМодуль is null)
      return d;
    foreach (путьПоПКНМодуля; d.дайПКНМодуля(папРазд))
```

```
    {
      auto importedModule = импортируйМодуль(путьПоПКНМодуля);
      if (importedModule is null)
        ошибка(d.начало, сооб.МодульНеЗагружен, путьПоПКНМодуля ~ ".d");
      модуль.модули ~= importedModule;
    }
    return d;
  }

  Д посети(ДекларацияАлиаса ad)
  {
    return ad;
  }

  Д посети(ДекларацияТипдефа td)
  {
    return td;
  }

  Д посети(ДекларацияПеречня d)
  {
    if (d.символ)
      return d;

    // Create the символ.
    d.символ = new Перечень(d.имя, d);

    бул анонимен_ли = d.символ.анонимен_ли;
    if (анонимен_ли)
      d.символ.имя = ТаблицаИд.генИДАнонПеречня();

    вставь(d.символ);

    auto parentScopeSymbol = масш.символ;
    auto enumSymbol = d.символ;
    войдиВМасштаб(d.символ);
    // Declare члены.
    foreach (член; d.члены)
    {
      посетиД(член);

      if (анонимен_ли) // Also вставь into родитель Масштаб if enum is
anonymous.
        вставь(член.символ, parentScopeSymbol);

      член.символ.тип = enumSymbol.тип; // Присвоить ТипПеречень.
    }
    выйдиИзМасштаба();
    return d;
  }

  Д посети(ДекларацияЧленаПеречня d)
  {
    d.символ = new ЧленПеречня(d.имя, защита, классХранения, типКомпоновки,
d);
    вставь(d.символ);
    return d;
  }

  Д посети(ДекларацияКласса d)
  {
    if (d.символ)
      return d;
```

```
  // Create the символ.
  d.символ = new Класс(d.имя, d);
  // Insert into current Масштаб.
  вставь(d.символ);
  войдиВМасштаб(d.символ);
  // Далее semantic analysis.
  d.деклы && посетиД(d.деклы);
  выйдиИзМасштаба();
  return d;
}

Д посети(ДекларацияИнтерфейса d)
{
  if (d.символ)
    return d;
  // Create the символ.
  d.символ = new drc.semantic.Symbols.Интерфейс(d.имя, d);
  // Insert into current Масштаб.
  вставь(d.символ);
  войдиВМасштаб(d.символ);
    // Далее semantic analysis.
    d.деклы && посетиД(d.деклы);
  выйдиИзМасштаба();
  return d;
}

Д посети(ДекларацияСтруктуры d)
{
  if (d.символ)
    return d;
  // Create the символ.
  d.символ = new Структура(d.имя, d);

  if (d.символ.анонимен_ли)
    d.символ.имя = ТаблицаИд.genAnonStructID();
  // Insert into current Масштаб.
  вставь(d.символ);

  войдиВМасштаб(d.символ);
    // Далее semantic analysis.
    d.деклы && посетиД(d.деклы);
  выйдиИзМасштаба();

  if (d.символ.анонимен_ли)
    // Insert члены into родитель Масштаб as well.
    foreach (член; d.символ.члены)
      вставь(член);
  return d;
}

Д посети(ДекларацияСоюза d)
{
  if (d.символ)
    return d;
  // Create the символ.
  d.символ = new Союз(d.имя, d);

  if (d.символ.анонимен_ли)
    d.символ.имя = ТаблицаИд.genAnonUnionID();

  // Insert into current Масштаб.
  вставь(d.символ);
```

```
    войдиВМасштаб(d.символ);
      // Далее semantic analysis.
      d.деклы && посетиД(d.деклы);
    выйдиИзМасштаба();

    if (d.символ.анонимен_ли)
      // Insert члены into родитель Масштаб as well.
      foreach (член; d.символ.члены)
        вставь(член);
    return d;
  }

  Д посети(ДекларацияКонструктора d)
  {
    auto func = new Функция(Идент.Ктор, d);
    вставьПерегрузку(func);
    return d;
  }

  Д посети(ДекларацияСтатическогоКонструктора d)
  {
    auto func = new Функция(Идент.Ктор, d);
    вставьПерегрузку(func);
    return d;
  }

  Д посети(ДекларацияДеструктора d)
  {
    auto func = new Функция(Идент.Дтор, d);
    вставьПерегрузку(func);
    return d;
  }

  Д посети(ДекларацияСтатическогоДеструктора d)
  {
    auto func = new Функция(Идент.Дтор, d);
    вставьПерегрузку(func);
    return d;
  }

  Д посети(ДекларацияФункции d)
  {
    auto func = new Функция(d.имя, d);
    вставьПерегрузку(func);
    return d;
  }

  Д посети(ДекларацияПеременных vd)
  {
    // Ошибка if we are in an interface.
    if (масш.символ.Интерфейс_ли && !vd.статический_ли)
      return ошибка(vd.начало, сооб.УИнтерфейсаНеДолжноБытьПеременных), vd;

    // Insert переменная символы in this declaration into the символ таблица.
    foreach (i, имя; vd.имена)
    {
      auto переменная = new Переменная(имя, защита, классХранения,
типКомпоновки, vd);
      переменная.значение = vd.инициализаторы[i];
      vd.переменные ~= переменная;
      вставь(переменная);
    }
    return vd;
```

```
}

Д посети(ДекларацияИнварианта d)
{
  auto func = new Функция(Идент.Инвариант, d);
  вставь(func);
  return d;
}

Д посети(ДекларацияЮниттеста d)
{
  auto func = new Функция(Идент.Юниттест, d);
  вставьПерегрузку(func);
  return d;
}

Д посети(ДекларацияОтладки d)
{
  if (d.определение_ли)
  { // debug = Id | Цел
    if (!масштабМодуля_ли())
      ошибка(d.начало, сооб.DebugSpecModuleLevel, d.спец.исхТекст);
    else if (d.спец.вид == ТОК.Идентификатор)
      контекст.добавьИдОтладки(d.спец.идент.ткт);
    else
      контекст.уровеньОтладки = d.спец.бцел_;
  }
  else
  { // debug ( Condition )
    if (выборОтладВетви(d.услов, контекст))
      d.компилированныеДеклы = d.деклы;
    else
      d.компилированныеДеклы = d.деклыИначе;
    d.компилированныеДеклы && посетиД(d.компилированныеДеклы);
  }
  return d;
}

Д посети(ДекларацияВерсии d)
{
  if (d.определение_ли)
  { // version = Id | Цел
    if (!масштабМодуля_ли())
      ошибка(d.начало, сооб.VersionSpecModuleLevel, d.спец.исхТекст);
    else if (d.спец.вид == ТОК.Идентификатор)
      контекст.добавьИдВерсии(d.спец.идент.ткт);
    else
      контекст.уровеньВерсии = d.спец.бцел_;
  }
  else
  { // version ( Condition )
    if (выборВерсионВетви(d.услов, контекст))
      d.компилированныеДеклы = d.деклы;
    else
      d.компилированныеДеклы = d.деклыИначе;
    d.компилированныеДеклы && посетиД(d.компилированныеДеклы);
  }
  return d;
}

Д посети(ДекларацияШаблона d)
{
  if (d.символ)
```

```
    return d;
  // Create the символ.
  d.символ = new Шаблон(d.имя, d);
  // Insert into current Масштаб.
  вставьПерегрузку(d.символ);
  return d;
}

Д посети(ДекларацияНов d)
{
  auto func = new Функция(Идент.Нов, d);
  вставь(func);
  return d;
}

Д посети(ДекларацияУдали d)
{
  auto func = new Функция(Идент.Удалить, d);
  вставь(func);
  return d;
}

// Attributes:

Д посети(ДекларацияЗащиты d)
{
  auto saved = защита; // Save.
  защита = d.защ; // Set.
  посетиД(d.деклы);
  защита = saved; // Restore.
  return d;
}

Д посети(ДекларацияКлассаХранения d)
{
  auto saved = классХранения; // Save.
  классХранения = d.классХранения; // Set.
  посетиД(d.деклы);
  классХранения = saved; // Restore.
  return d;
}

Д посети(ДекларацияКомпоновки d)
{
  auto saved = типКомпоновки; // Save.
  типКомпоновки = d.типКомпоновки; // Set.
  посетиД(d.деклы);
  типКомпоновки = saved; // Restore.
  return d;
}

Д посети(ДекларацияРазложи d)
{
  auto saved = размерРаскладки; // Save.
  размерРаскладки = d.размер; // Set.
  посетиД(d.деклы);
  размерРаскладки = saved; // Restore.
  return d;
}

Д посети(ДекларацияСтатическогоПодтверди d)
{
  return d;
```

```
  }

  Д посети(ДекларацияСтатическогоЕсли d)
  {
    return d;
  }

  Д посети(ДекларацияСмеси d)
  {
    return d;
  }

  Д посети(ДекларацияПрагмы d)
  {
    if (d.идент is Идент.сооб)
    {
      // TODO
    }
    else
    {
      семантикаПрагмы(масш, d.начало, d.идент, d.арги);
      посетиД(d.деклы);
    }
    return d;
  }
} // override


/+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  |                                  Statements
|

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+/

  /// The current surrounding, breakable statement.
  И breakableStatement;

  И setBS(И s)
  {
    auto old = breakableStatement;
    breakableStatement = s;
    return old;
  }

  проц  restoreBS(И s)
  {
    breakableStatement = s;
  }

override
{
  И посети(СложнаяИнструкция s)
  {
    foreach (stmnt; s.инстрции)
      посетиИ(stmnt);
    return s;
  }

  И посети(НелегальнаяИнструкция)
  { assert(0, "semantic pass on invalid AST"); return null; }

  И посети(ПустаяИнструкция s)
  {
```

```
    return s;
  }

  И посети(ИнструкцияТелаФункции s)
  {
    return s;
  }

  И посети(ИнструкцияМасштаб s)
  {
//      войдиВМасштаб();
    посетиИ(s.s);
//      выйдиИзМасштаба();
    return s;
  }

  И посети(ИнструкцияСМеткой s)
  {
    return s;
  }

  И посети(ИнструкцияВыражение s)
  {
    return s;
  }

  И посети(ИнструкцияДекларация s)
  {
    return s;
  }

  И посети(ИнструкцияЕсли s)
  {
    return s;
  }

  И посети(ИнструкцияПока s)
  {
    auto saved = setBS(s);
    // TODO:
    restoreBS(saved);
    return s;
  }

  И посети(ИнструкцияДелайПока s)
  {
    auto saved = setBS(s);
    // TODO:
    restoreBS(saved);
    return s;
  }

  И посети(ИнструкцияПри s)
  {
    auto saved = setBS(s);
    // TODO:
    restoreBS(saved);
    return s;
  }

  И посети(ИнструкцияСКаждым s)
  {
    auto saved = setBS(s);
```

```
  // TODO:
  // find overload opApply or opApplyReverse.
  restoreBS(saved);
  return s;
}

// D2.0
И посети(ИнструкцияДиапазонСКаждым s)
{
  auto saved = setBS(s);
  // TODO:
  restoreBS(saved);
  return s;
}

И посети(ИнструкцияЩит s)
{
  auto saved = setBS(s);
  // TODO:
  restoreBS(saved);
  return s;
}

И посети(ИнструкцияРеле s)
{
  auto saved = setBS(s);
  // TODO:
  restoreBS(saved);
  return s;
}

И посети(ИнструкцияДефолт s)
{
  auto saved = setBS(s);
  // TODO:
  restoreBS(saved);
  return s;
}

И посети(ИнструкцияДалее s)
{
  return s;
}

И посети(ИнструкцияВсё s)
{
  return s;
}

И посети(ИнструкцияИтог s)
{
  return s;
}

И посети(ИнструкцияПереход s)
{
  return s;
}

И посети(ИнструкцияДля s)
{
  return s;
}
```

```
И посети(ИнструкцияСинхр s)
{
  return s;
}

И посети(ИнструкцияПробуй s)
{
  return s;
}

И посети(ИнструкцияЛови s)
{
  return s;
}

И посети(ИнструкцияИтожь s)
{
  return s;
}

И посети(ИнструкцияСтражМасштаба s)
{
  return s;
}

И посети(ИнструкцияБрось s)
{
  return s;
}

И посети(ИнструкцияЛетучее s)
{
  return s;
}

И посети(ИнструкцияБлокАсм s)
{
  foreach (stmnt; s.инструкции.инстрции)
    посетиИ(stmnt);
  return s;
}

И посети(ИнструкцияАсм s)
{
  return s;
}

И посети(ИнструкцияАсмРасклад s)
{
  return s;
}

И посети(ИнструкцияНелегальныйАсм)
{ assert(0, "semantic pass on invalid AST"); return null; }

И посети(ИнструкцияПрагма s)
{
  return s;
}

И посети(ИнструкцияСмесь s)
{
```

```
    return s;
  }

  И посети(ИнструкцияСтатическоеЕсли s)
  {
    return s;
  }

  И посети(ИнструкцияСтатическоеПодтверди s)
  {
    return s;
  }

  И посети(ИнструкцияОтладка s)
  {
    return s;
  }

  И посети(ИнструкцияВерсия s)
  {
    return s;
  }
} // override


/+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  |                                 Expressions
|

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+/

  /// Determines whether в issue an ошибка when a символ couldn't be found.
  бул errorOnUndefinedSymbol;
  //бул errorOnUnknownSymbol;

  /// Reports an ошибка if 'e' is of тип бул.
  проц  errorЕслиBool(Выражение в)
  {
    ошибка(в.начало, "the operation is not defined for the тип бул");
  }

  /// Returns a call выражение if 'e' overrides
  /// an operatorwith the имя 'ид'.
  /// Параметры:
  ///   в = the binary выражение в be checked.
  ///   ид = the имя of the overload function.
  Выражение найдиПерегрузку(УнарноеВыражение в, Идентификатор* ид)
  {
    // TODO:
    // check в for struct or class
    // ищи for function named ид
    // return call выражение: в.opXYZ()
    return null;
  }

  /// Returns a call выражение if 'e' overrides
  /// an operator with the имя 'ид' or 'id_r'.
  /// Параметры:
  ///   в = the binary выражение в be checked.
  ///   ид = the имя of the overload function.
  ///   id_r = the имя of the reverse overload function.
  Выражение найдиПерегрузку(БинарноеВыражение в, Идентификатор* ид,
Идентификатор* id_r)
```

```
  {
    // TODO:
    return null;
  }

override
{
  В посети(НелегальноеВыражение)
  { assert(0, "semantic pass on invalid AST"); return null; }

  В посети(ВыражениеУсловия в)
  {
    return в;
  }

  В посети(ВыражениеЗапятая в)
  {
    if (!в.естьТип)
    {
      в.лв = посетиВ(в.лв);
      в.пв = посетиВ(в.пв);
      в.тип = в.пв.тип;
    }
    return в;
  }

  В посети(ВыражениеИлиИли в)
  {
    return в;
  }

  В посети(ВыражениеИИ в)
  {
    return в;
  }

  В посети(ВыражениеИли в)
  {
    if (auto о = найдиПерегрузку(в, Идент.opOr, Идент.opOr_r))
      return о;
    return в;
  }

  В посети(ВыражениеИИли в)
  {
    if (auto о = найдиПерегрузку(в, Идент.opXor, Идент.opXor_r))
      return о;
    return в;
  }

  В посети(ВыражениеИ в)
  {
    if (auto о = найдиПерегрузку(в, Идент.opAnd, Идент.opAnd_r))
      return о;
    return в;
  }

  В посети(ВыражениеРавно в)
  {
    if (auto о = найдиПерегрузку(в, Идент.opEquals, null))
      return о;
    return в;
  }
```

```
В посети(ВыражениеРавенство в)
{
  return в;
}

В посети(ВыражениеОтнош в)
{
  if (auto о = найдиПерегрузку(в, Идент.opCmp, null))
    return о;
  return в;
}

В посети(ВыражениеВхо в)
{
  if (auto о = найдиПерегрузку(в, Идент.opIn, Идент.opIn_r))
    return о;
  return в;
}

В посети(ВыражениеЛСдвиг в)
{
  if (auto о = найдиПерегрузку(в, Идент.opShl, Идент.opShl_r))
    return о;
  return в;
}

В посети(ВыражениеПСдвиг в)
{
  if (auto о = найдиПерегрузку(в, Идент.opShr, Идент.opShr_r))
    return о;
  return в;
}

В посети(ВыражениеБПСдвиг в)
{
  if (auto о = найдиПерегрузку(в, Идент.opUShr, Идент.opUShr_r))
    return о;
  return в;
}

В посети(ВыражениеПлюс в)
{
  if (auto о = найдиПерегрузку(в, Идент.opAdd, Идент.opAdd_r))
    return о;
  return в;
}

В посети(ВыражениеМинус в)
{
  if (auto о = найдиПерегрузку(в, Идент.opSub, Идент.opSub_r))
    return о;
  return в;
}

В посети(ВыражениеСоедини в)
{
  if (auto о = найдиПерегрузку(в, Идент.opCat, Идент.opCat_r))
    return о;
  return в;
}

В посети(ВыражениеУмножь в)
```

```
{
  if (auto o = найдиПерегрузку(в, Идент.opMul, Идент.opMul_r))
    return o;
  return в;
}

В посети(ВыражениеДели в)
{
  if (auto o = найдиПерегрузку(в, Идент.opDiv, Идент.opDiv_r))
    return o;
  return в;
}

В посети(ВыражениеМод в)
{
  if (auto o = найдиПерегрузку(в, Идент.opMod, Идент.opMod_r))
    return o;
  return в;
}

В посети(ВыражениеПрисвой в)
{
  if (auto o = найдиПерегрузку(в, Идент.opAssign, null))
    return o;
  // TODO: also check for opIndexAssign and opSliceAssign.
  return в;
}

В посети(ВыражениеПрисвойЛСдвиг в)
{
  if (auto o = найдиПерегрузку(в, Идент.opShlAssign, null))
    return o;
  return в;
}

В посети(ВыражениеПрисвойПСдвиг в)
{
  if (auto o = найдиПерегрузку(в, Идент.opShrAssign, null))
    return o;
  return в;
}

В посети(ВыражениеПрисвойБПСдвиг в)
{
  if (auto o = найдиПерегрузку(в, Идент.opUShrAssign, null))
    return o;
  return в;
}

В посети(ВыражениеПрисвойИли в)
{
  if (auto o = найдиПерегрузку(в, Идент.opOrAssign, null))
    return o;
  return в;
}

В посети(ВыражениеПрисвойИ в)
{
  if (auto o = найдиПерегрузку(в, Идент.opAndAssign, null))
    return o;
  return в;
}
```

```
В посети(ВыражениеПрисвойПлюс в)
{
  if (auto о = найдиПерегрузку(в, Идент.opAddAssign, null))
    return о;
  return в;
}

В посети(ВыражениеПрисвойМинус в)
{
  if (auto о = найдиПерегрузку(в, Идент.opSubAssign, null))
    return о;
  return в;
}

В посети(ВыражениеПрисвойДел в)
{
  auto о = найдиПерегрузку(в, Идент.opDivAssign, null);
  if (о)
    return о;
  return в;
}

В посети(ВыражениеПрисвойУмн в)
{
  auto о = найдиПерегрузку(в, Идент.opMulAssign, null);
  if (о)
    return о;
  return в;
}

В посети(ВыражениеПрисвойМод в)
{
  auto о = найдиПерегрузку(в, Идент.opModAssign, null);
  if (о)
    return о;
  return в;
}

В посети(ВыражениеПрисвойИИли в)
{
  auto о = найдиПерегрузку(в, Идент.opXorAssign, null);
  if (о)
    return о;
  return в;
}

В посети(ВыражениеПрисвойСоед в)
{
  auto о = найдиПерегрузку(в, Идент.opCatAssign, null);
  if (о)
    return о;
  return в;
}

В посети(ВыражениеАдрес в)
{
  if (в.естьТип)
    return в;
  в.в = посетиВ(в.в);
  в.тип = в.в.тип.укНа();
  return в;
}
```

```
В посети(ВыражениеПреИнкр в)
{
  if (в.естьТип)
    return в;
  // TODO: rewrite в в+=1
  в.в = посетиВ(в.в);
  в.тип = в.в.тип;
  errorЕслиBool(в.в);
  return в;
}

В посети(ВыражениеПреДекр в)
{
  if (в.естьТип)
    return в;
  // TODO: rewrite в в-=1
  в.в = посетиВ(в.в);
  в.тип = в.в.тип;
  errorЕслиBool(в.в);
  return в;
}

В посети(ВыражениеПостИнкр в)
{
  if (в.естьТип)
    return в;
  if (auto о = найдиПерегрузку(в, Идент.opPostInc))
    return о;
  в.в = посетиВ(в.в);
  в.тип = в.в.тип;
  errorЕслиBool(в.в);
  return в;
}

В посети(ВыражениеПостДекр в)
{
  if (в.естьТип)
    return в;
  if (auto о = найдиПерегрузку(в, Идент.opPostDec))
    return о;
  в.в = посетиВ(в.в);
  в.тип = в.в.тип;
  errorЕслиBool(в.в);
  return в;
}

В посети(ВыражениеДереф в)
{
  if (в.естьТип)
    return в;
version(D2)
  if (auto о = найдиПерегрузку(в, Идент.opStar))
    return о;
  в.в = посетиВ(в.в);
  в.тип = в.в.тип.следщ;
  if (!в.в.тип.указатель_ли)
  {
    ошибка(в.в.начало,
      "dereference operator '*x' not defined for выражение of тип '{}'",
      в.в.тип.вТкст());
    в.тип = Типы.Ошибка;
  }
  // TODO:
```

```
    // if (в.в.тип.isVoid)
    //   ошибка();
    return в;
  }

  В посети(ВыражениеЗнак в)
  {
    if (в.естьТип)
      return в;
    if (auto о = найдиПерегрузку(в, в.отриц_ли ? Идент.opNeg : Идент.opPos))
      return о;
    в.в = посетиВ(в.в);
    в.тип = в.в.тип;
    errorЕслиBool(в.в);
    return в;
  }

  В посети(ВыражениеНе в)
  {
    if (в.естьТип)
      return в;
    в.в = посетиВ(в.в);
    в.тип = Типы.Бул;
    // TODO: в.в must be convertible в бул.
    return в;
  }

  В посети(ВыражениеКомп в)
  {
    if (в.естьТип)
      return в;
    if (auto о = найдиПерегрузку(в, Идент.opCom))
      return о;
    в.в = посетиВ(в.в);
    в.тип = в.в.тип;
    if (в.тип.плавающий_ли || в.тип.бул_ли)
    {
      ошибка(в.начало, "ОПЕРАТОР '~x' не определён для типа '{}'",
в.тип.вТкст());
      в.тип = Типы.Ошибка;
    }
    return в;
  }

  В посети(ВыражениеВызов в)
  {
    if (auto о = найдиПерегрузку(в, Идент.opCall))
      return о;
    return в;
  }

  В посети(ВыражениеНов в)
  {
    return в;
  }

  В посети(ВыражениеНовАнонКласс в)
  {
    return в;
  }

  В посети(ВыражениеУдали в)
  {
```

```
  return в;
}

В посети(ВыражениеКаст в)
{
  if (auto о = найдиПерегрузку(в, Идент.opCast))
    return о;
  return в;
}

В посети(ВыражениеИндекс в)
{
  if (auto о = найдиПерегрузку(в, Идент.opIndex))
    return о;
  return в;
}

В посети(ВыражениеСрез в)
{
  if (auto о = найдиПерегрузку(в, Идент.opSlice))
    return о;
  return в;
}

В посети(ВыражениеТочка в)
{
  if (в.естьТип)
    return в;
  бул resetIdScope = идМасштаб is null;
  // TODO:
  resetIdScope && (идМасштаб = null);
  return в;
}

В посети(ВыражениеМасштабМодуля в)
{
  if (в.естьТип)
    return в;
  бул resetIdScope = идМасштаб is null;
  идМасштаб = модуль;
  в.в = посетиВ(в.в);
  в.тип = в.в.тип;
  resetIdScope && (идМасштаб = null);
  return в;
}

В посети(ВыражениеИдентификатор в)
{
  if (в.естьТип)
    return в;
  debug(sema) выдай.форматнс("", в);
  auto идСема = в.идСема();
  в.символ = ищи(идСема);
  return в;
}

В посети(ВыражениеЭкземплярШаблона в)
{
  if (в.естьТип)
    return в;
  debug(sema) выдай.форматнс("", в);
  auto идСема = в.идСема();
  в.символ = ищи(идСема);
```

```
  return в;
}

В посети(ВыражениеСпецСема в)
{
  if (в.естьТип)
    return в.значение;
  switch (в.особаяСема.вид)
  {
  case ТОК.СТРОКА, ТОК.ВЕРСИЯ:
    в.значение = new ЦелВыражение(в.особаяСема.бцел_, Типы.Бцел);
    break;
  case ТОК.ФАЙЛ, ТОК.ДАТА, ТОК.ВРЕМЯ, ТОК.ШТАМПВРЕМЕНИ, ТОК.ПОСТАВЩИК:
    в.значение = new ТекстовоеВыражение(в.особаяСема.ткт);
    break;
  default:
    assert(0);
  }
  в.тип = в.значение.тип;
  return в.значение;
}

В посети(ВыражениеЭтот в)
{
  return в;
}

В посети(ВыражениеСупер в)
{
  return в;
}

В посети(ВыражениеНуль в)
{
  if (!в.естьТип)
    в.тип = Типы.Проц_ук;
  return в;
}

В посети(ВыражениеДоллар в)
{
  if (в.естьТип)
    return в;
  в.тип = Типы.Т_мера;
  // if (!inArraySubscript)
  //   ошибка("$ can only be in an массив subscript.");
  return в;
}

В посети(БулевоВыражение в)
{
  assert(в.естьТип);
  return в.значение;
}

В посети(ЦелВыражение в)
{
  if (в.естьТип)
    return в;

  if (в.число & 0x8000_0000_0000_0000)
    в.тип = Типы.Бдол; // 0xFFFF_FFFF_FFFF_FFFF
  else if (в.число & 0xFFFF_FFFF_0000_0000)
```

```
    в.тип = Типы.Дол; // 0x7FFF_FFFF_FFFF_FFFF
  else if (в.число & 0x8000_0000)
    в.тип = Типы.Бцел; // 0xFFFF_FFFF
  else
    в.тип = Типы.Цел; // 0x7FFF_FFFF
  return в;
}

В посети(ВыражениеРеал в)
{
  if (!в.естьТип)
    в.тип = Типы.Дво;
  return в;
}

В посети(ВыражениеКомплекс в)
{
  if (!в.естьТип)
    в.тип = Типы.Кдво;
  return в;
}

В посети(ВыражениеСим в)
{
  assert(в.естьТип);
  return в.значение;
}

В посети(ТекстовоеВыражение в)
{
  assert(в.естьТип);
  return в;
}

В посети(ВыражениеЛитералМассива в)
{
  return в;
}

В посети(ВыражениеЛитералАМассива в)
{
  return в;
}

В посети(ВыражениеПодтверди в)
{
  return в;
}

В посети(ВыражениеСмесь в)
{
  return в;
}

В посети(ВыражениеИмпорта в)
{
  return в;
}

В посети(ВыражениеТипа в)
{
  return в;
}
```

```
В посети(ВыражениеИдТипаТочка в)
{
  return в;
}

В посети(ВыражениеИдТипа в)
{
  return в;
}

В посети(ВыражениеЯвляется в)
{
  return в;
}

В посети(ВыражениеРодит в)
{
  if (!в.естьТип)
  {
    в.следщ = посетиВ(в.следщ);
    в.тип = в.следщ.тип;
  }
  return в;
}

В посети(ВыражениеЛитералФункции в)
{
  return в;
}

В посети(ВыражениеТрактовки в) // D2.0
{
  return в;
}

В посети(ВыражениеИницПроц в)
{
  return в;
}

В посети(ВыражениеИницМассива в)
{
  return в;
}

В посети(ВыражениеИницСтруктуры в)
{
  return в;
}

В посети(ВыражениеТипАсм в)
{
  return в;
}

В посети(ВыражениеСмещениеАсм в)
{
  return в;
}

В посети(ВыражениеСегАсм в)
{
```

```
    return в;
  }

  В посети(ВыражениеАсмПослеСкобки в)
  {
    return в;
  }

  В посети(ВыражениеАсмСкобка в)
  {
    return в;
  }

  В посети(ВыражениеЛокальногоРазмераАсм в)
  {
    return в;
  }

  В посети(ВыражениеАсмРегистр в)
  {
    return в;
  }
} // override


/+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  |                                    Типы
|

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+/

override
{
  Т посети(НелегальныйТип)
  { assert(0, "semantic pass on invalid AST"); return null; }

  Т посети(ИнтегральныйТип т)
  {
    // A таблица mapping the вид of a сема в its corresponding semantic Тип.
    ТипБазовый[ТОК] семВТип = [
      ТОК.Сим : Типы.Сим,   ТОК.Шим : Типы.Шим,   ТОК.Дим : Типы.Дим, ТОК.Бул
: Типы.Бул,
      ТОК.Байт : Типы.Байт,   ТОК.Ббайт : Типы.Ббайт,   ТОК.Крат : Типы.Крат,
ТОК.Бкрат : Типы.Бкрат,
      ТОК.Цел : Типы.Цел,    ТОК.Бцел : Типы.Бцел,    ТОК.Дол : Типы.Дол,
ТОК.Бдол : Типы.Бдол,
      ТОК.Цент : Типы.Цент,   ТОК.Бцент : Типы.Бцент,
      ТОК.Плав : Типы.Плав,  ТОК.Дво : Типы.Дво,  ТОК.Реал : Типы.Реал,
      ТОК.Вплав : Типы.Вплав, ТОК.Вдво : Типы.Вдво, ТОК.Вреал : Типы.Вреал,
      ТОК.Кплав : Типы.Кплав, ТОК.Кдво : Типы.Кдво, ТОК.Креал : Типы.Креал,
ТОК.Проц : Типы.Проц
    ];
    assert(т.лекс in семВТип);
    т.тип = семВТип[т.лекс];
    return т;
  }

  Т посети(КвалифицированныйТип т)
  {
    // Reset идМасштаб at the конец if this the корень КвалифицированныйТип.
    бул resetIdScope = идМасштаб is null;
//     if (т.лв.Является!(КвалифицированныйТип) is null)
//       идМасштаб = null; // Reset at левый-most тип.
```

```
    посетиТ(т.лв);
    // Присвоить the символ of the левый-hand сторона в идМасштаб.
    setIdScope(т.лв.символ);
    посетиТ(т.пв);
//      setIdScope(т.пв.символ);
    // Присвоить члены of the правый-hand сторона в this тип.
    т.тип = т.пв.тип;
    т.символ = т.пв.символ;
    // Reset идМасштаб.
    resetIdScope && (идМасштаб = null);
    return т;
  }

  Т посети(ТМасштабМодуля т)
  {
    идМасштаб = модуль;
    return т;
  }

  Т посети(ТИдентификатор т)
  {
    auto идСема = т.начало;
    auto символ = ищи(идСема);
    // TODO: save символ or its тип in т.
    return т;
  }

  Т посети(ТТип т)
  {
    т.в = посетиВ(т.в);
    т.тип = т.в.тип;
    return т;
  }

  Т посети(ТЭкземплярШаблона т)
  {
    auto идСема = т.начало;
    auto символ = ищи(идСема);
    // TODO: save символ or its тип in т.
    return т;
  }

  Т посети(ТУказатель т)
  {
    т.тип = посетиТ(т.следщ).тип.укНа();
    return т;
  }

  Т посети(ТМассив т)
  {
    auto типОснова = посетиТ(т.следщ).тип;
    if (т.ассоциативный_ли)
      т.тип = типОснова.массивИз(посетиТ(т.ассоцТип).тип);
    else if (т.динамический_ли)
      т.тип = типОснова.массивИз();
    else if (т.статический_ли)
    {}
    else
      assert(т.срез_ли);
    return т;
  }

  Т посети(ТФункция т)
```

```
  {
    return t;
  }

  T посети(ТДелегат t)
  {
    return t;
  }

  T посети(ТУказательНаФункСи t)
  {
    return t;
  }

  T посети(ТипКлассОснова t)
  {
    return t;
  }

  T посети(ТКонст t) // D2.0
  {
    return t;
  }

  T посети(ТИнвариант t) // D2.0
  {
    return t;
  }
} // override


/+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  |                                  Параметры
|

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+/

override
{
  У посети(Параметр p)
  {
    return p;
  }

  У посети(Параметры p)
  {
    return p;
  }

  У посети(ПараметрАлиасШаблона p)
  {
    return p;
  }

  У посети(ПараметрТипаШаблона p)
  {
    return p;
  }

  У посети(ПараметрЭтотШаблона p) // D2.0
  {
    return p;
  }
```

```
  У посети(ПараметрШаблонЗначения р)
  {
    return р;
  }

  У посети(ПараметрКортежШаблона р)
  {
    return р;
  }

  У посети(ПараметрыШаблона р)
  {
    return р;
  }

  У посети(АргументыШаблона р)
  {
    return р;
  }
} // override
}
```

module drc.semantic.Scope;

```
import drc.semantic.Symbol,
       drc.semantic.Symbols;
import drc.lexer.Identifier;
import common;


/// Выполняет построение иерархии сред.
class Масштаб
{
  Масштаб родитель; /// Охватывающий Масштаб, или null, если this является
корневым Масштабом.

  СимволМасштаба символ; /// Текущий символ.

  this(Масштаб родитель, СимволМасштаба символ);

  /// Найти символ в данном Масштабе.
  /// Параметры:
  ///   имя = название символа.
  Символ сыщи(Идентификатор* имя);

  /// Ищет символ в данном Масштабе и во всех охватывающих.
  /// Параметры:
  ///   имя = название символа.
  Символ ищи(Идентификатор* имя);

  /// Ищет символ в данном Масштабе и во всех охватывающих.
  /// Параметры:
  ///   имя = название символа.
  ///   ignoreSymbol = символ, который следует пропустить.
  Символ ищи(Идентификатор* имя, Символ ignoreSymbol);

  /// Создаёт новый внутренний масштаб и возвращает его.
  Масштаб войдиВ(СимволМасштаба символ);

  ///Разрушает текущий Масштаб и возвращает внешний Масштаб.
  Масштаб выход();
```

```
  /// Находит Масштаб включающего Класса.
  Масштаб масштабКласса();

  /// Находит Масштаб включающего Модуля.
  Масштаб масштабМодуля();
}
```

module drc.semantic.Symbol;

```
import drc.ast.Node;
import drc.lexer.Identifier;
import common;

/// Перечень ИДов символов.
enum СИМ
{
  Модуль,
  Пакет,
  Класс,
  Интерфейс,
  Структура,
  Союз,
  Перечень,
  ЧленПеречня,
  Шаблон,
  Переменная,
  Функция,
  Алиас,
  НаборПерегрузки,
  Масштаб,
//   Тип,
}

/// Символ представляет собой объект с информации о семантике кода.
class Символ
{ /// Перечень состояний символа.
  enum Состояние : бкрат
  {
    Объявлен,   /// Символ был декларирован.
    Обрабатывается, /// Символ обрабатывается.
    Обработан    /// Символ обработан.
  }

  СИМ сид; /// ИД данного символа.
  Состояние состояние; /// Семантическое состояние данного символа.
  Символ родитель; /// Родитель, к которому относится данный символ.
  Идентификатор* имя; /// Название символа.
  /// Узел синтактического дерева, произвёдший данный символ.
  /// Useful for source код положение инфо and retrioцени of doc comments.
  Узел узел;

  /// Строит Символ объект.
  /// Параметры:
  ///   сид = the символ's ID.
  ///   имя = the символ's имя.
  ///   узел = the символ's узел.
  this(СИМ сид, Идентификатор* имя, Узел узел)
  {
    this.сид = сид;
    this.имя = имя;
    this.узел = узел;
  }
```

```
  /// Change the состояние в Состояние.Обрабатывается.
  проц  устОбрабатывается()
  { состояние = Состояние.Обрабатывается; }

  /// Change the состояние в Состояние.Обработан.
  проц  устОбработан()
  { состояние = Состояние.Обработан; }

  /// Returns да if the символ is being completed.
  бул обрабатывается_ли()
  { return состояние == Состояние.Обрабатывается; }

  /// Returns да if the символы is complete.
  бул обработан_ли()
  { return состояние == Состояние.Обработан; }

  /// A template macro for building isXYZ() methods.
  private template isX(ткст вид)
  {
    const ткст isX = `бул `~вид~`_ли(){ return сид == СИМ.`~вид~`; }`;
  }
  mixin(isX!("Модуль"));
  mixin(isX!("Пакет"));
  mixin(isX!("Класс"));
  mixin(isX!("Интерфейс"));
  mixin(isX!("Структура"));
  mixin(isX!("Союз"));
  mixin(isX!("Перечень"));
  mixin(isX!("ЧленПеречня"));
  mixin(isX!("Шаблон"));
  mixin(isX!("Переменная"));
  mixin(isX!("Функция"));
  mixin(isX!("Алиас"));
  mixin(isX!("НаборПерегрузки"));
  mixin(isX!("Масштаб"));
//   mixin(isX!("Тип"));

  /// Casts the символ в Класс.
  Класс в(Класс)()
  {
    assert(mixin(`this.сид == mixin("СИМ." ~ Класс.stringof)`));
    return cast(Класс)cast(ук)this;
  }

  /// Возвращает: the fully qualified имя of this символ.
  /// E.g.: drc.semantic.Symbol.Символ.дайПКН
  ткст дайПКН()
  {
    if (!имя)
      return родитель ? родитель.дайПКН() : "";
    if (родитель)
      return родитель.дайПКН() ~ '.' ~ имя.ткт;
    return имя.ткт;
  }
}
```

module drc.semantic.SymbolTable;

```
import drc.semantic.Symbol;
import drc.lexer.Identifier;
import common;

/// Помещает идентификатор типа ткст в Символ.
```

```
struct ТаблицаСимволов
{
  Символ[сим[]] таблица; /// Структура таблицы данных.

  /// Ищет идент в таблице.
  /// Возвращает: символ, если он там имеется, либо null.
  Символ сыщи(Идентификатор* идент)
  {
    assert(идент !is null);
    auto psym = идент.ткт in таблица;
    return psym ? *psym : null;
  }

  /// Вставляет символ в таблицу.
  проц  вставь(Символ символ, Идентификатор* идент)
  {
    таблица[идент.ткт] = символ;
  }
}
```

module drc.semantic.Types;

```
import drc.semantic.Symbol,
       drc.semantic.TypesEnum;
import drc.lexer.Identifier;
import drc.CompilerInfo;

import common;

/// Базовый тип для всех типовых структур.
abstract class Тип/* : Символ*/
{
  Тип следщ;     /// Следующий тип в структуре типов.
  ТИП тид;       /// ИД типа.
  Символ символ; /// Не null, если у типа есть символ.

  this(){}

  /// Строит Тип объект.
  /// Параметры:
  ///   следщ = следщ тип.
  ///   тид = Ид типа
  this(Тип следщ, ТИП тид)
  {
//     this.сид = СИМ.Тип;

    this.следщ = следщ;
    this.тид = тид;
  }

  /// Returns да if this тип equals the другой one.
  бул opEquals(Тип другой)
  {
    // TODO:
    return нет;
  }

  /// Returns a pointer тип в this тип.
  УказательТип укНа()
  {
    return new УказательТип(this);
  }
```

```
/// Returns a dynamic массив тип using this тип as its base.
ДМассивТип массивИз()
{
  return new ДМассивТип(this);
}

/// Returns an associative массив тип using this тип as its base.
/// Параметры:
///   key = the key тип.
АМассивТип массивИз(Тип key)
{
  return new АМассивТип(this, key);
}

/// Возвращает байт размер of this тип.
final т_мера размера()
{
  return МИТаблица.дайРазмер(this);
}

/// Size is not in МИТаблица. Find out via virtual method.
т_мера sizeOf_()
{
  return размера();
}

/// Returns да if this тип has a символ.
бул естьСимвол_ли()
{
  return символ !is null;
}

/// Возвращает тип as a ткст.
abstract ткст вТкст();

/// Returns да if this тип is a бул тип.
бул бул_ли()
{
  return тид == ТИП.Бул;
}

/// Returns да if this тип is a pointer тип.
бул указатель_ли()
{
  return тид == ТИП.Указатель;
}

/// Returns да if this тип is an integral число тип.
бул интегральный_ли()
{
  switch (тид)
  {
  case ТИП.Сим, ТИП.Шим, ТИП.Дим, ТИП.Бул, ТИП.Байт, ТИП.Ббайт,
       ТИП.Крат, ТИП.Бкрат, ТИП.Цел, ТИП.Бцел, ТИП.Дол, ТИП.Бдол,
       ТИП.Цент, ТИП.Бцент:
    return да;
  default:
    return нет;
  }
}

/// Returns да if this тип is a floating point число тип.
бул плавающий_ли()
```

```
  {
    return реал_ли() || воображаемый_ли() || комплексный_ли();
  }

  /// Returns да if this тип is a реал число тип.
  бул реал_ли()
  {
    return тид == ТИП.Плав || тид == ТИП.Дво || тид == ТИП.Реал;
  }

  /// Returns да if this тип is an imaginary число тип.
  бул воображаемый_ли()
  {
    return тид == ТИП.Вплав || тид == ТИП.Вдво || тид == ТИП.Вреал;
  }

  /// Returns да if this тип is a complex число тип.
  бул комплексный_ли()
  {
    return тид == ТИП.Кплав || тид == ТИП.Кдво || тид == ТИП.Креал;
  }
}

/// All basic types. E.g.: цел, сим, реал etc.
class ТипБазовый : Тип
{
  this(ТИП typ)
  {
    super(null, typ);
  }

  ткст вТкст()
  {
    return [
      ТИП.Сим : "сим"[], ТИП.Шим : "шим", ТИП.Дим : "дим",
      ТИП.Бул : "бул", ТИП.Байт : "байт", ТИП.Ббайт : "ббайт",
      ТИП.Крат : "крат", ТИП.Бкрат : "бкрат", ТИП.Цел : "цел",
      ТИП.Бцел : "бцел", ТИП.Дол : "дол", ТИП.Бдол : "бдол",
      ТИП.Цент : "цент", ТИП.Бцент : "бцент", ТИП.Плав : "плав",
      ТИП.Дво : "дво", ТИП.Реал : "реал", ТИП.Вплав : "вплав",
      ТИП.Вдво : "вдво", ТИП.Вреал : "вреал", ТИП.Кплав : "кплав",
      ТИП.Кдво : "кдво", ТИП.Креал : "креал"
    ][this.тид];
  }
}

/// Dynamic массив тип.
class ДМассивТип : Тип
{
  this(Тип следщ)
  {
    super(следщ, ТИП.ДМассив);
  }

  ткст вТкст()
  {
    return следщ.вТкст() ~ "[]";
  }
}

/// Associative массив тип.
class АМассивТип : Тип
{
```

```
  Тип клТип;
  this(Тип следщ, Тип клТип)
  {
    super(следщ, ТИП.АМассив);
    this.клТип = клТип;
  }

  ткст вТкст()
  {
    return следщ.вТкст() ~ "[" ~ клТип.вТкст() ~ "]";
  }
}

/// Статический массив тип.
class СМассивТип : Тип
{
  т_мера dimension;
  this(Тип следщ, т_мера dimension)
  {
    super(следщ, ТИП.СМассив);
    this.dimension = dimension;
  }

  ткст вТкст()
  {
    return Формат("%s[%d]", следщ.вТкст(), dimension);
  }
}

/// Указатель тип.
class УказательТип : Тип
{
  this(Тип следщ)
  {
    super(следщ, ТИП.Указатель);
  }

  ткст вТкст()
  {
    return следщ.вТкст() ~ "*";
  }
}

/// Ссылка тип.
class ТСсылка : Тип
{
  this(Тип следщ)
  {
    super(следщ, ТИП.Ссылка);
  }

  ткст вТкст()
  { // FIXME: this is probably wrong.
    return следщ.вТкст() ~ "&";
  }
}

/// Перечень тип.
class ПереченьТип : Тип
{
  this(Символ символ)
  {
    super(типОснова, ТИП.Перечень);
```

```
    this.символ = символ;
  }

  /// Setter for the base тип.
  проц  типОснова(Тип тип)
  {
    следщ = тип;
  }

  /// Getter for the base тип.
  Тип типОснова()
  {
    return следщ;
  }

  ткст вТкст()
  {
    return символ.имя.ткт;
  }
}

/// Структура тип.
class TСтруктура : Тип
{
  this(Символ символ)
  {
    super(null, ТИП.Структура);
    this.символ = символ;
  }

  ткст вТкст()
  {
    return символ.имя.ткт;
  }
}

/// Класс тип.
class ТКласс : Тип
{
  this(Символ символ)
  {
    super(null, ТИП.Класс);
    this.символ = символ;
  }

  ткст вТкст()
  {
    return символ.имя.ткт;
  }
}

/// Типдеф тип.
class ТипдефТип : Тип
{
  this(Тип следщ)
  {
    super(следщ, ТИП.Типдеф);
  }

  ткст вТкст()
  { // TODO:
    return "типдеф";
  }
```

```
}

/// Функция тип.
class ФункцияТип : Тип
{
  this(Тип следщ)
  {
    super(следщ, ТИП.Функция);
  }

  ткст вТкст()
  { // TODO:
    return "функция";
  }
}

/// Делегат тип.
class ДелегатТип : Тип
{
  this(Тип следщ)
  {
    super(следщ, ТИП.Делегат);
  }

  ткст вТкст()
  { // TODO:
    return "делегат";
  }
}

/// Идентификатор тип.
class ИдентификаторТип : Тип
{
  Идентификатор* идент;
  this(Идентификатор* идент)
  {
    super(null, ТИП.Идентификатор);
  }

  ткст вТкст()
  {
    return идент.ткт;
  }
}

/// Шаблон instantiation тип.
class ЭкземплШаблонаТип : Тип
{
  this()
  {
    super(null, ТИП.ШЭкземпляр);
  }

  ткст вТкст()
  { // TODO:
    return "шабл!()";
  }
}

/// Шаблон tuple тип.
class КортежТип : Тип
{
  this(Тип следщ)
```

```
  {
    super(следщ, ТИП.Кортеж);
  }

  ткст вТкст()
  { // TODO:
    return "кортеж";
  }
}

/// Constant тип. D2.0
class КонстантаТип : Тип
{
  this(Тип следщ)
  {
    super(следщ, ТИП.Конст);
  }

  ткст вТкст()
  {
    return "конст(" ~ следщ.вТкст() ~ ")";
  }
}

/// Инвариант тип. D2.0
class ИнвариантТип : Тип
{
  this(Тип следщ)
  {
    super(следщ, ТИП.Конст);
  }

  ткст вТкст()
  {
    return "инвариант(" ~ следщ.вТкст() ~ ")";
  }
}

/// Represents a значение related в a Тип.
union Значение
{
  ук  упроц;
  бул   бул_;
  дим  дим_;
  дол   дол_;
  бдол  бдол_;
  цел    цел_;
  бцел   бцел_;
  плав  плав_;
  дво дво_;
  реал   реал_;
  креал  креал_;
}

/// Информация related в a Тип.
struct МетаИнфоТип
{
  сим mangle; /// Mangle символ of the тип.
  бкрат размер; /// Байт размер of the тип.
  Значение* дефолтИниц; /// Дефолт initialization значение.
}

/// Namespace for the meta инфо таблица.
```

```
struct МИТаблица
{
static:
  const бкрат РАЗМЕР_НЕ_ДОСТУПЕН = 0; /// Size not available.
  const Значение ЗНОЛЬ = {цел_:0}; /// Значение 0.
  const Значение ЗНУЛЬ = {упроц:null}; /// Значение null.
  const Значение V0xFF = {дим_:0xFF}; /// Значение 0xFF.
  const Значение V0xFFFF = {дим_:0xFFFF}; /// Значение 0xFFFF.
  const Значение ЗЛОЖЬ = {бул_:нет}; /// Значение нет.
  const Значение ЗНЕЧ = {плав_:плав.nan}; /// Значение NAN.
  const Значение ЗКНЕЧ = {креал_:креал.nan}; /// Значение complex NAN.
  private alias РАЗМЕР_НЕ_ДОСТУПЕН РНД;
  private alias РАЗМЕР_УК РА;
  /// The meta инфо таблица.
  private const МетаИнфоТип метаИнфоТаблица[] = [
    {'?', РНД}, // Ошибка

    {'a', 1, &V0xFF},   // Сим
    {'u', 2, &V0xFFFF},   // Шим
    {'w', 4, &V0xFFFF},   // Дим
    {'b', 1, &ЗЛОЖЬ},   // Бул
    {'g', 1, &ЗНОЛЬ},   // Байт
    {'h', 1, &ЗНОЛЬ},   // Ббайт
    {'s', 2, &ЗНОЛЬ},   // Крат
    {'t', 2, &ЗНОЛЬ},   // Бкрат
    {'i', 4, &ЗНОЛЬ},   // Цел
    {'k', 4, &ЗНОЛЬ},   // Бцел
    {'l', 8, &ЗНОЛЬ},   // Дол
    {'m', 8, &ЗНОЛЬ},   // Бдол
    {'?', 16, &ЗНОЛЬ},  // Цент
    {'?', 16, &ЗНОЛЬ},  // Бцент
    {'f', 4, &ЗНЕЧ},   // Плав
    {'d', 8, &ЗНЕЧ},   // Дво
    {'e', 12, &ЗНЕЧ},  // Реал
    {'o', 4, &ЗНЕЧ},   // Вплав
    {'p', 8, &ЗНЕЧ},   // Вдво
    {'j', 12, &ЗНЕЧ},  // Вреал
    {'q', 8, &ЗКНЕЧ},   // Кплав
    {'r', 16, &ЗКНЕЧ},  // Кдво
    {'c', 24, &ЗКНЕЧ},  // Креал
    {'v', 1},   // проц

    {'n', РНД},  // Нет

    {'A', РА*2, &ЗНУЛЬ}, // Dynamic массив
    {'G', РА*2, &ЗНУЛЬ}, // Статический массив
    {'H', РА*2, &ЗНУЛЬ}, // Associative массив

    {'E', РНД}, // Перечень
    {'S', РНД}, // Структура
    {'C', РА, &ЗНУЛЬ},  // Класс
    {'T', РНД}, // Типдеф
    {'F', РА},  // Функция
    {'D', РА*2, &ЗНУЛЬ}, // Делегат
    {'P', РА, &ЗНУЛЬ},  // Указатель
    {'R', РА, &ЗНУЛЬ},  // Ссылка
    {'I', РНД}, // Идентификатор
    {'?', РНД}, // Шаблон instance
    {'B', РНД}, // Кортеж
    {'x', РНД}, // Конст, D2
    {'y', РНД}, // Инвариант, D2
  ];
  static assert(метаИнфоТаблица.length == ТИП.max+1);
```

```
  /// Возвращает размер of a тип.
  т_мера дайРазмер(Тип тип)
  {
    auto размер = метаИнфоТаблица[тип.тид].размер;
    if (размер == РАЗМЕР_НЕ_ДОСТУПЕН)
      return тип.sizeOf_();
    return размер;
  }
}

/// Namespace for a установи of predefined types.
struct Типы
{
static:
  /// Predefined basic types.
  ТипБазовый Сим,   Шим,   Дим, Бул,
             Байт,   Ббайт,   Крат, Бкрат,
             Цел,    Бцел,    Дол,  Бдол,
             Цент,   Бцент,
             Плав,  Дво,  Реал,
             Вплав, Вдво, Вреал,
             Кплав, Кдво, Креал, Проц;

  ТипБазовый Т_мера; /// The размер тип.
  ТипБазовый Т_дельтаук; /// The pointer difference тип.
  УказательТип Проц_ук; /// The проц  pointer тип.
  ТипБазовый Ошибка; /// The ошибка тип.
  ТипБазовый Неопределённый; /// The undefined тип.
  ТипБазовый ПокаНеИзвестен; /// The символ is undefined but might be
resolved.

  /// Allocates an instance of ТипБазовый and assigns it в имяТипа.
  template новТБ(ткст имяТипа)
  {
    const новТБ = mixin(имяТипа~" = new ТипБазовый(ТИП."~имяТипа~")");
  }

  /// Initializes predefined types.
  static this()
  {
    новТБ!("Сим");
    новТБ!("Шим");
    новТБ!("Дим");
    новТБ!("Бул");
    новТБ!("Байт");
    новТБ!("Ббайт");
    новТБ!("Крат");
    новТБ!("Бкрат");
    новТБ!("Цел");
    новТБ!("Бцел");
    новТБ!("Дол");
    новТБ!("Бдол");
    новТБ!("Цент");
    новТБ!("Бцент");
    новТБ!("Плав");
    новТБ!("Дво");
    новТБ!("Реал");
    новТБ!("Вплав");
    новТБ!("Вдво");
    новТБ!("Вреал");
    новТБ!("Кплав");
    новТБ!("Кдво");
```

```
    новТБ!("Креал");
    новТБ!("Проц");
    version(X86_64)
    {
      Т_мера = Бдол;
      Т_дельтаук = Дол;
    }
    else
    {
      Т_мера = Бцел;
      Т_дельтаук = Цел;
    }
    Проц_ук = Проц.укНа;
    Ошибка = new ТипБазовый(ТИП.Ошибка);
    Неопределённый = new ТипБазовый(ТИП.Ошибка);
    ПокаНеИзвестен = new ТипБазовый(ТИП.Ошибка);
  }
}
```

module drc.semantic.TypesEnum;

```
/// Перечень идентификаторов типов.
enum ТИП
{
  Ошибка,
  // Basic types.
  Сим,     /// сим
  Шим,    /// шим
  Дим,    /// дим
  Бул,     /// бул
  Байт,     /// int8
  Ббайт,    /// uint8
  Крат,    /// int16
  Бкрат,   /// uint16
  Цел,      /// int32
  Бцел,     /// uint32
  Дол,     /// int64
  Бдол,    /// uint64
  Цент,     /// int128
  Бцент,    /// uint128
  Плав,    /// float32
  Дво,   /// float64
  Реал,     /// float80
  Вплав,   /// imaginary float32
  Вдво, /// imaginary float64
  Вреал,    /// imaginary float80
  Кплав,   /// complex float32
  Кдво, /// complex float64
  Креал,    /// complex float80
  Проц,     /// проц

  Нет,    /// TypeNone in the specs. Why?

  ДМассив, /// Динамический массив.
  СМассив, /// Статический массив.
  АМассив, /// Ассоциативный массив.

  Перечень,        /// An enum.
  Структура,      /// A struct.
  Класс,       /// A class.
  Типдеф,     /// A typedef.
  Функция,    /// A function.
  Делегат,    /// A delegate.
```

```
  Указатель,    /// A pointer.
  Ссылка,  /// A reference.
  Идентификатор, /// An identifier.
  ШЭкземпляр,  /// Шаблон instance.
  Кортеж,      /// A template tuple.
  Конст,      /// A constant тип. D2.0
  Инвариант,  /// An invariant тип. D2.0
}
```

Лексер (lexer)

```
module drc.lexer.Funcs;

import drc.Unicode : юАльфа_ли;

const дим _Z_ = 26; /// Control+Z.

/// Возвращает: да, если d является разделителем строк или абзацев Unicode.
бул симНовСтрЮ_ли(дим d)
{
  return d == РСд || d == РАд;
}

/// Возвращает: да, если p указывает на разделитель строки или абзаца.
бул новСтрЮ_ли(сим* p)
{
  return *p == РС[0] && p[1] == РС[1] && (p[2] == РС[2] || p[2] == РА[2]);
}

/// Возвращает: да, если p указывает на начало новой строки Новстр.
/// $(PRE
/// Новстр := "\n" | "\r" | "\r\n" | РС | РА
/// РС := "\u2028"
/// РА := "\u2029"
/// )
бул новСтр_ли(сим* p)
{
  return *p == '\n' || *p == '\r' || новСтрЮ_ли(p);
}

/// Возвращает: да, если c это Новстр символ.
бул новСтр_ли(дим c)
{
  return c == '\n' || c == '\r' || симНовСтрЮ_ли(c);
}

/// Возвращает: да, если p указывает на КФ символ (конец файла).
/// $(PRE
/// КФ := "\0" | _Z_
/// _Z_ := "\x1A"
/// )
бул кф_ли(дим c)
{
  return c == 0 || c == _Z_;
}

/// Возвращает: да, если p указывает на первый символ EndOfLine.
/// $(PRE EndOfLine := Новстр | КФ)
бул конецСтроки_ли(сим* p)
{
  return новСтр_ли(p) || кф_ли(*p);
}
```

```
/// Сканирует символ Новстр и устанавливает р на символ после него.
/// Возвращает: да, если он найден или нет в прот.сл.
бул сканируйНовСтр(ref сим* р)
in { assert(р); }
body
{
  switch (*р)
  {
  case '\r':
    if (р[1] == '\n')
      ++р;
  case '\n':
    ++р;
    break;
  default:
    if (новСтрЮ_ли(р))
      р += 3;
    else
      return нет;
  }
  return да;
}

/// Сканирует символ Новстр и устанавливает р на символ после него.
/// Возвращает: да, если он найден или нет в прот.сл.
бул сканируйНовСтр(ref сим* р, сим* конец)
in { assert(р && р < конец); }
body
{
  switch (*р)
  {
  case '\r':
    if (р+1 < конец && р[1] == '\n')
      ++р;
  case '\n':
    ++р;
    break;
  default:
    if (р+2 < конец && новСтрЮ_ли(р))
      р += 3;
    else
      return нет;
  }
  return да;
}

/// Сканирует Новстр в обрантом направлении и устанавливает конец
/// на первый символ нс.
/// Возвращает: да, если он найден или нет в прот.сл.
бул сканируйНовСтрРеверс(сим* начало, ref сим* конец)
{
  switch (*конец)
  {
  case '\n':
    if (начало <= конец-1 && конец[-1] == '\r')
      конец--;
  case '\r':
    break;
  case РС[2], РА[2]:
    if (начало <= конец-2 && конец[-1] == РС[1] && конец[-2] == РС[0]) {
      конец -= 2;
      break;
```

```
    }
  // fall through
  default:
    return нет;
  }
  return да;
}

/// Сканирует идентификатор D.
/// Параметры:
///   ref_p = откуда начать.
///   конец = где закончить.
/// Возвращает: идентификатор, если он действителен (устанавливает ref_p на 1
после ид,) или
///             null, если недействителен (оставляет неизменным ref_p.)
ткст сканируйИдентификатор(ref сим* ref_p, сим* конец)
in { assert(ref_p && ref_p < конец); }
body
{
  auto p = ref_p;
  if (начсим_ли(*p) || юАльфа_ли(p, конец)) // IdStart
  {
    do // IdChar*
      p++;
    while (p < конец && (идент_ли(*p) || юАльфа_ли(p, конец)))
    auto identifier = ref_p[0 .. p-ref_p];
    ref_p = p;
    return identifier;
  }
  return null;
}

/// Таблица свойств символов ASCII.
static const цел ptable[256] = [
 0, 0, 0, 0, 0, 0, 0, 0, 0,32, 0,32,32, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
32, 0, 0x2200, 0, 0, 0, 0, 0x2700, 0, 0, 0, 0, 0, 0, 0, 0,
 7, 7, 7, 7, 7, 7, 7, 7, 6, 6, 0, 0, 0, 0, 0, 0x3f00,
 0,12,12,12,12,12,12, 8, 8, 8, 8, 8, 8, 8, 8, 8,
 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 0x5c00, 0, 0,16,
 0, 0x70c, 0x80c,12,12,12, 0xc0c, 8, 8, 8, 8, 8, 8, 8, 0xa08, 8,
 8, 8, 0xd08, 8, 0x908, 8, 0xb08, 8, 8, 8, 8, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
];

/// Перечень флагов свойств символов.
enum СвойствоС
{
       Восмиричный = 1,    /// 0-7
       Десятичный = 1<<1, /// 0-9
         Гекс = 1<<2, /// 0-9a-fA-F
       Буква = 1<<3, /// a-zA-Z
  Подчерк = 1<<4, /// _
  Пробельный = 1<<5  /// ' ' \t \v \f
}
```

```
const бцел EVMask = 0xFF00; // Bit mask for escape значение.

private alias СвойствоС СР;
/// Возвращает: да if c is an octal digit.
цел восмир_ли(сим c) { return ptable[c] & СР.Восмиричный; }
/// Возвращает: да if c is a decimal digit.
цел цифра_ли(сим c) { return ptable[c] & СР.Десятичный; }
/// Возвращает: да if c is a hexadecimal digit.
цел гекс_ли(сим c) { return ptable[c] & СР.Гекс; }
/// Возвращает: да if c is a letter.
цел буква_ли(сим c) { return ptable[c] & СР.Буква; }
/// Возвращает: да if c is an alphanumeric.
цел цифробукв_ли(сим c) { return ptable[c] & (СР.Буква | СР.Десятичный); }
/// Возвращает: да if c is the beginning of a D identifier (only ASCII.)
цел начсим_ли(сим c) { return ptable[c] & (СР.Буква | СР.Подчерк); }
/// Возвращает: да if c is a D identifier символ (only ASCII.)
цел идент_ли(сим c) { return ptable[c] & (СР.Буква | СР.Подчерк |
СР.Десятичный); }
/// Возвращает: да if c is a whitespace символ.
цел пбел_ли(сим c) { return ptable[c] & СР.Пробельный; }
/// Возвращает: the escape значение for c.
цел сим8еск(сим c) { return ptable[c] >> 8; /*(ptable[c] & EVMask) >> 8;*/ }
/// Возвращает: да if c is an ASCII символ.
цел аски_ли(бцел c) { return c < 128; }

version(gen_ptable)
static this()
{
  alias ptable p;
  assert(p.length == 256);
  // Initialize символ properties таблица.
  for (цел i; i < p.length; ++i)
  {
    p[i] = 0; // Reset
    if ('0' <= i && i <= '7')
      p[i] |= СР.Восмиричный;
    if ('0' <= i && i <= '9')
      p[i] |= СР.Десятичный | СР.Гекс;
    if ('a' <= i && i <= 'f' || 'A' <= i && i <= 'F')
      p[i] |= СР.Гекс;
    if ('a' <= i && i <= 'z' || 'A' <= i && i <= 'Z')
      p[i] |= СР.Буква;
    if (i == '_')
      p[i] |= СР.Подчерк;
    if (i == ' ' || i == '\t' || i == '\v' || i == '\f')
      p[i] |= СР.Пробельный;
  }
  // Store escape sequence значения in second байт.
  assert(СвойствоС.max <= ббайт.max, "символ property флаги and escape
значение байт overlap.");
  p['\''] |= 39 << 8;
  p['"'] |= 34 << 8;
  p['?'] |= 63 << 8;
  p['\\'] |= 92 << 8;
  p['a'] |= 7 << 8;
  p['b'] |= 8 << 8;
  p['f'] |= 12 << 8;
  p['n'] |= 10 << 8;
  p['r'] |= 13 << 8;
  p['t'] |= 9 << 8;
  p['v'] |= 11 << 8;
  // Print a formatted массив literal.
  ткст массив = "[\n";
```

```
  foreach (i, c; ptable)
  {
    массив ~= Формат((c>255?" 0x{0:x},":"{0,2},"), c) ~ (((i+1) % 16) ?
"":"\n");
  }
  массив[$-2..$] = "\n]";
  выдай(массив).нс;
}
```

module drc.lexer.Identifier;

```
import drc.lexer.TokensEnum,
       drc.lexer.IdentsEnum;
import common;

/// Представляет идентификатор по определению в спецификации D.
///
/// $(PRE
///  Идентификатор := НачалоИд СимвИд*
///  НачалоИд := "_" | Буква
///  СимвИд := НачалоИд | "0"-"9"
///  Буква := ЮАльфа
/// )
/// See_Also:
///  Алфавитные символы Unicode определены в Unicode 5.0.0.
align(1)
struct Идентификатор
{
  ткст ткт; /// Идентификатор в ткст UTF-8.
  ТОК вид;   /// Вид семы.
  ВИД видИд; /// Только для предопределённых идентификаторов.

  static Идентификатор* opCall(ткст ткт, ТОК вид)
  {
    auto ид = new Идентификатор;
    ид.ткт = ткт;
    ид.вид = вид;
    return ид;
  }

  static Идентификатор* opCall(ткст ткт, ТОК вид, ВИД видИд)
  {
    auto ид = new Идентификатор;
    ид.ткт = ткт;
    ид.вид = вид;
    ид.видИд = видИд;
    return ид;
  }

  бцел вХэш()
  {
    бцел хэш;
    foreach(c; ткт) {
      хэш *= 11;
      хэш += c;
    }
    return хэш;
  }
}
// pragma(сооб, Идентификатор.sizeof.stringof);
```

module drc.lexer.IdentsEnum;

```
import drc.lexer.IdentsGenerator;

version(DDoc)
  enum ВИД : бкрат; /// Перечень видов предопределенных идентификаторов.
else
mixin(
  // Перечисляет предопределённые идентификаторы.
  "enum ВИД : бкрат {"
    "Нуль,"
    ~ генерируйЧленыИД ~
  "}"
);
```

---

module drc.lexer.IdentsGenerator;

```
/// Таблица предопределенных идентификаторов.
///
/// Формат ('#' начинает комментарии):
/// $(PRE
/// ПредопределенныйИдентификатор := ИмяИсхКода (":" ТекстИда)?
/// ИмяИсхКода := Идентификатор # Имя, которое будет использоваться в
исходном коде.
/// ТекстИда := Пусто | Идентификатор # Действительный текст идентификатора.
/// Пусто := ""                    # ТекстИда может быть пустым.
/// Идентификатор := см. модуль $(MODLINK drc.lexer.Identifier).
/// )
/// Если ТекстИда не указан, то дефолтом является ИмяИсхКода.
private static const сим[][] предопрИденты = [
  // Специальный пустой идентификатор:
  "Пусто:",
  // Предопределенные идентификаторы версии:
  "DigitalMars", "X86", "X86_64",
  /*"Windows", */"Win32", "Win64",
  "Linux:linux", "ЛитлЭндиан", "BigEndian",
  "D_Coverage", "D_InlineAsm_X86", "D_Version2",
  "none", "all",
  // Вариадические параметры:
  "Аргументы:_arguments", "Аргук:_argptr",
  // масштаб(Идентификатор):
  "выход", "успех", "сбой", "exit", "success", "failure",
  // Прагмы:
  "сооб", "lib", "startaddress", "msg",
  // Конпоновка:
  "C", "D", "Windows", "Pascal", "System",
  // Con-/Destructor:
  "Ктор:__ctor", "Дтор:__dtor",
  // new() and delete() methods.
  "Нов:__new", "Удалить:__delete",
  // Юниттест and invariant.
  "Юниттест:__unittest", "Инвариант:__invariant",
  // Методы перегрузки операторов
  "opNeg", "opPos", "opCom",
  "opEquals", "opCmp",  "opAssign",
  "opAdd",  "opAdd_r",  "opAddAssign",
  "opSub",  "opSub_r",  "opSubAssign",
  "opMul",  "opMul_r",  "opMulAssign",
  "opDiv",  "opDiv_r",  "opDivAssign",
  "opMod",  "opMod_r",  "opModAssign",
  "opAnd",  "opAnd_r",  "opAndAssign",
  "opOr",   "opOr_r",   "opOrAssign",
  "opXor",  "opXor_r",  "opXorAssign",
  "opShl",  "opShl_r",  "opShlAssign",
```

```
  "opShr",  "opShr_r",  "opShrAssign",
  "opUShr", "opUShr_r", "opUShrAssign",
  "opCat",  "opCat_r",  "opCatAssign",
  "opIn",   "opIn_r",
  "opIndex", "opIndexAssign",
  "opSlice", "opSliceAssign",
  "opPostInc",
  "opPostDec",
  "opCall",
  "opCast",
  "opStar", // D2
  // foreach and foreach_reverse:
  "opApply", "opApplyReverse",
  // Entry function:
  "main",
  // ASM identifiers:
  "near", "far", "word", "dword", "qword",
  "ptr", "offset", "seg", "__LOCAL_SIZE",
  "FS", "ST",
  "AL", "AH", "AX", "EAX",
  "BL", "BH", "BX", "EBX",
  "CL", "CH", "CX", "ECX",
  "DL", "DH", "DX", "EDX",
  "BP", "EBP", "SP", "ESP",
  "DI", "EDI", "SI", "ESI",
  "ES", "CS", "SS", "DS", "GS",
  "CR0", "CR2", "CR3", "CR4",
  "DR0", "DR1", "DR2", "DR3", "DR6", "DR7",
  "TR3", "TR4", "TR5", "TR6", "TR7",
  "MM0", "MM1", "MM2", "MM3",
  "MM4", "MM5", "MM6", "MM7",
  "XMM0", "XMM1", "XMM2", "XMM3",
  "XMM4", "XMM5", "XMM6", "XMM7",
];

сим[][] дайПару(ткст текстИда)
{
  foreach (i, c; текстИда)
    if (c == ':')
      return [текстИда[0..i], текстИда[i+1..текстИда.length]];
  return [текстИда, текстИда];
}

unittest
{
  static assert(
    дайПару("тест") == ["тест", "тест"] &&
    дайПару("тест:tset") == ["тест", "tset"] &&
    дайПару("empty:") == ["empty", ""]
  );
}

/++
  CTF для генерации членов структуры Идент.

  Результирующий текст выглядить примерно так:
  ---
  private struct Иды {static const:
    Идентификатор _Empty = {"", ТОК.Идентификатор, ВИД.Пусто};
    Идентификатор _main = {"main", ТОК.Идентификатор, ВИД.main};
    // etc.
  }
  Идентификатор* Пусто = &Иды._Empty;
```

```
  Идентификатор* main = &Иды._main;
  // etc.
  private Идентификатор*[] __allIds = [
    Пусто,
    main,
    // и т.д.
  ];
  ---
+/
ткст генерируйЧленыИдент()
{
  ткст приват_члены = "private struct Иды {static const:";
  ткст публ_члены = "";
  ткст массив = "private Идентификатор*[] __allIds = [";

  foreach (идент; предопрИденты)
  {
    сим[][] пара = дайПару(идент);
    // Идентификатор _name = {"имя", ТОК.Идентификатор, ID.имя};
    приват_члены ~= "Идентификатор _"~пара[0]~` = {"`~пара[1]~`",
ТОК.Идентификатор, ВИД.`~пара[0]~"};\n";
    // Идентификатор* имя = &_name;
    публ_члены ~= "Идентификатор* "~пара[0]~" = &Иды._"~пара[0]~";\n";
    массив ~= пара[0]~",";
  }

  приват_члены ~= "}"; // Close private {
  массив ~= "];";

  return приват_члены ~ публ_члены ~ массив;
}

/// CTF for generating the члены of the enum ВИД.
ткст генерируйЧленыИД()
{
  ткст члены;
  foreach (идент; предопрИденты)
    члены ~= дайПару(идент)[0] ~ ",\n";
  return члены;
}

// pragma(сооб, генерируйЧленыИдент());
// pragma(сооб, генерируйЧленыИД());
```

module drc.lexer.IdTable;

```
import drc.lexer.TokensEnum,
       drc.lexer.IdentsGenerator,
       drc.lexer.Keywords;
import common;

public import drc.lexer.Identifier,
              drc.lexer.IdentsEnum;

/// Неймспейс для предопределенных идентификаторов.
struct Идент
{
  const static
  {
    mixin(генерируйЧленыИдент());
  }

  /// Возвращает массив предопределенных идентификаторов.
```

```
  static Идентификатор*[] всеИды()
  {
    return __allIds;
  }
}

/// Глобальная таблица для размещения и получения идентификаторов.
struct ТаблицаИд
{
static:
  /// Набор общих, предопределенных идентификаторов для быстрых поисков.
  private Идентификатор*[ткст] статическаяТаблица;
  /// Таблица, растущая с каждым новым уникальным идентификатором.
  private Идентификатор*[ткст] растущаяТаблица;

  /// Загружает ключевые слова и предопределенные идентификаторы в
статическую таблицу.
  static this()
  {
    foreach (ref k; g_reservedIds)
      статическаяТаблица[k.ткт] = &k;
    foreach (ид; Идент.всеИды())
      статическаяТаблица[ид.ткт] = ид;
    статическаяТаблица.rehash;
  }

  /// Ищет ткстИда в обеих таблицах.
  Идентификатор* сыщи(ткст ткстИда)
  {
    auto ид = вСтатической(ткстИда);
    if (ид)
      return ид;
    return вРастущей(ткстИда);
  }

  /// Ищет ткстИда в статической таблице.
  Идентификатор* вСтатической(ткст ткстИда)
  {
    auto ид = ткстИда in статическаяТаблица;
    return ид ? *ид : null;
  }

  alias Идентификатор* function(ткст ткстИда) ФункцияПоиска;
  /// Ищет ткстИда в растущей таблице.
  ФункцияПоиска вРастущей = &_inGrowing_unsafe; // Дефолт на небезопасную
функцию.

  /// Устанавливает режим безопасности нити для растущей таблицы.
  проц  установиНитебезопасность(бул b)
  {
    if (b)
      вРастущей = &_inGrowing_safe;
    else
      вРастущей = &_inGrowing_unsafe;
  }

  /// Возвращает да, если доступ к растущей таблице нитебезопасен.
  бул нитебезопасно_ли()
  {
    return вРастущей is &_inGrowing_safe;
  }

  /// Ищет ткстИда в таблице.
```

```
///
/// Добавляет ткстИда в таблицу, если он не найден.
private Идентификатор* _inGrowing_unsafe(ткст ткстИда)
out(ид)
{ assert(ид !is null); }
body
{
  auto ид = ткстИда in растущаяТаблица;
  if (ид)
    return *ид;
  auto newID = Идентификатор(ткстИда, ТОК.Идентификатор);
  растущаяТаблица[ткстИда] = newID;
  return newID;
}

/// Ищет ткстИда в таблица.
///
/// Добавляет ткстИда в таблицу, если не найден.
/// Доступ в структуру данных синхронизирован.
private Идентификатор* _inGrowing_safe(ткст ткстИда)
{
  synchronized
    return _inGrowing_unsafe(ткстИда);
}

/+
Идентификатор* addIdentifiers(сим[][] idStrings)
{
  auto ids = new Идентификатор*[idStrings.length];
  foreach (i, ткстИда; idStrings)
  {
    Идентификатор** ид = ткстИда in tabulatedIds;
    if (!ид)
    {
      auto newID = Идентификатор(ТОК.Идентификатор, ткстИда);
      tabulatedIds[ткстИда] = newID;
      ид = &newID;
    }
    ids[i] = *ид;
  }
}
+/

static бцел anonCount; /// Счётчик бля безымянных идентификаторов.

/// Генерирует безымянный идентификатор.
///
/// Конкатенирует префикс с anonCount.
/// Этот идентификатор не помещён в таблицу.
Идентификатор* генБезымянныйИД (ткст prefix)
{
  ++anonCount;
  auto x = anonCount;
  // Convert счёт в a ткст and добавь it в ткт.
  ткст чис;
  do
    чис = cast(сим)('0' + (x % 10)) ~ чис;
  while (x /= 10)
  return Идентификатор(prefix ~ чис, ТОК.Идентификатор);
}

/// Генерирует идентификатор для безымянного перечня.
Идентификатор* генИДАнонПеречня()
```

```
  {
    return генБезымянныйИД ("__anonenum");
  }

  /// Генерирует идентификатор для безымянного класса.
  Идентификатор* genAnonClassID()
  {
    return генБезымянныйИД ("__anonclass");
  }

  /// Генерирует идентификатор для безымянной структуры.
  Идентификатор* genAnonStructID()
  {
    return генБезымянныйИД ("__anonstruct");
  }

  /// Генерирует идентификатор для анонимного союза.
  Идентификатор* genAnonUnionID()
  {
    return генБезымянныйИД ("__anonunion");
  }

  /// Генерирует идентификатор для модуля, у которого не указано имя.
  Идентификатор* генИдМодуля()
  {
    return генБезымянныйИД ("__module");
  }
}

unittest
{
  // TODO: пиши benchmark.
  // Single таблица

  // Single таблица. synchronized

  // Two tables.

  // Two tables. synchronized
}
```

module drc.lexer.Keywords;

```
import drc.lexer.Token,
       drc.lexer.Identifier;

/// Таблица резервированных идентификаторов.
static const Идентификатор[] g_reservedIds = [
  {"abstract", ТОК.Абстрактный},
  {"абстрактный", ТОК.Абстрактный},
  {"alias", ТОК.Алиас},
  {"иной", ТОК.Алиас},
  {"align", ТОК.Расклад},
  {"расклад", ТОК.Расклад},
  {"asm", ТОК.Асм},
  {"асм", ТОК.Асм},
  {"assert", ТОК.Подтвердить},
  {"подтверди", ТОК.Подтвердить},
  {"auto", ТОК.Авто},
  {"авто", ТОК.Авто},
  {"body", ТОК.Тело},
  {"тело", ТОК.Тело},
  {"bool", ТОК.Бул},
```

```
{"бул", ТОК.Бул},
{"break", ТОК.Всё},
{"всё", ТОК.Всё},
{"byte", ТОК.Байт},
{"байт", ТОК.Байт},
{"case", ТОК.Реле},
{"реле", ТОК.Реле},
{"cast", ТОК.Каст},
{"catch", ТОК.Кэтч},
{"cdouble", ТОК.Кдво},
{"кдво", ТОК.Кдво},
{"cent", ТОК.Цент},
{"цент", ТОК.Цент},
{"cfloat", ТОК.Кплав},
{"кплав", ТОК.Кплав},
{"char", ТОК.Сим},
{"сим", ТОК.Сим},
{"class", ТОК.Класс},
{"класс", ТОК.Класс},
{"const", ТОК.Конст},
{"конст", ТОК.Конст},
{"continue", ТОК.Далее},
{"далее", ТОК.Далее},
{"creal", ТОК.Креал},
{"креал", ТОК.Креал},
{"dchar", ТОК.Дим},
{"дим", ТОК.Дим},
{"debug", ТОК.Отладка},
{"отладка", ТОК.Отладка},
{"default", ТОК.Дефолт},
{"дефолт", ТОК.Дефолт},
{"delegate", ТОК.Делегат},
{"делегат", ТОК.Делегат},
{"delete", ТОК.Удалить},
{"удали", ТОК.Удалить},
{"deprecated", ТОК.Устаревший},
{"устаревший", ТОК.Устаревший},
{"do", ТОК.Делай},
{"делай", ТОК.Делай},
{"double", ТОК.Дво},
{"дво", ТОК.Дво},
{"else", ТОК.Иначе},
{"иначе", ТОК.Иначе},
{"enum", ТОК.Перечень},
{"перечень", ТОК.Перечень},
{"export", ТОК.Экспорт},
{"экспорт", ТОК.Экспорт},
{"extern", ТОК.Экстерн},
{"экстерн", ТОК.Экстерн},
{"false", ТОК.Ложь},
{"нет", ТОК.Ложь},
{"final", ТОК.Окончательный},
{"окончательный", ТОК.Окончательный},
{"finally", ТОК.Finally},
{"наконец", ТОК.Finally},
{"float", ТОК.Плав},
{"плав", ТОК.Плав},
{"for", ТОК.При},
{"при", ТОК.При},
{"foreach", ТОК.Длявсех},
{"длявсех", ТОК.Длявсех},
{"foreach_reverse", ТОК.Длявсех_реверс},
{"длявсехрев", ТОК.Длявсех_реверс},
```

```
{"function", ТОК.Функция},
{"функ", ТОК.Функция},
{"goto", ТОК.Переход},
{"переход_на", ТОК.Переход},
{"idouble", ТОК.Вдво},
{"вдво", ТОК.Вдво},
{"if", ТОК.Если},
{"если", ТОК.Если},
{"ifloat", ТОК.Вплав},
{"вплав", ТОК.Вплав},
{"import", ТОК.Импорт},
{"импорт", ТОК.Импорт},
{"in", ТОК.Вхо},
{"вхо", ТОК.Вхо},
{"inout", ТОК.Вховых},
{"вховых", ТОК.Вховых},
{"int", ТОК.Цел},
{"цел", ТОК.Цел},
{"interface", ТОК.Интерфейс},
{"интерфейс", ТОК.Интерфейс},
{"invariant", ТОК.Инвариант},
{"инвариант", ТОК.Инвариант},
{"ireal", ТОК.Вреал},
{"вреал", ТОК.Вреал},
{"is", ТОК.Является},
{"есть", ТОК.Является},
{"lazy", ТОК.Отложенный},
{"отложеный", ТОК.Отложенный},
{"long", ТОК.Дол},
{"дол", ТОК.Дол},
{"macro", ТОК.Макрос}, // D2.0
{"макро", ТОК.Макрос},
{"mixin", ТОК.Смесь},
{"впиши", ТОК.Смесь},
{"module", ТОК.Модуль},
{"модуль", ТОК.Модуль},
{"new", ТОК.Нов},
{"нов", ТОК.Нов},
{"nothrow", ТОК.Nothrow}, // D2.0
{"null", ТОК.Нуль},
{"пусто", ТОК.Нуль},
{"out", ТОК.Вых},
{"вых", ТОК.Вых},
{"override", ТОК.Перепись},
{"перепись", ТОК.Перепись},
{"package", ТОК.Пакет},
{"пакет", ТОК.Пакет},
{"pragma", ТОК.Прагма},
{"прагма", ТОК.Прагма},
{"private", ТОК.Приватный},
{"protected", ТОК.Защищённый},
{"public", ТОК.Публичный},
{"pure", ТОК.Pure}, // D2.0
{"real", ТОК.Реал},
{"реал", ТОК.Реал},
{"ref", ТОК.Реф},
{"return", ТОК.Итог},
{"итог", ТОК.Итог},
{"scope", ТОК.Масштаб},
{"short", ТОК.Крат},
{"крат", ТОК.Крат},
{"static", ТОК.Статический},
{"struct", ТОК.Структура},
```

```
    {"структ", ТОК.Структура},
    {"super", ТОК.Супер},
    {"супер", ТОК.Супер},
    {"switch", ТОК.Щит},
    {"щит", ТОК.Щит},
    {"synchronized", ТОК.Синхронизованный},
    {"синхронно", ТОК.Синхронизованный},
    {"template", ТОК.Шаблон},
    {"шаблон", ТОК.Шаблон},
    {"this", ТОК.Этот},
    {"этот", ТОК.Этот},
    {"throw", ТОК.Брось},
    {"брось", ТОК.Брось},
    {"__traits", ТОК.Трэтс}, // D2.0
    {"true", ТОК.Истина},
    {"да", ТОК.Истина},
    {"try", ТОК.Пробуй},
    {"пробуй", ТОК.Пробуй},
    {"typedef", ТОК.Типдеф},
    {"typeid", ТОК.Идтипа},
    {"typeof", ТОК.Типа},
    {"ubyte", ТОК.Ббайт},
    {"ббайт", ТОК.Ббайт},
    {"ucent", ТОК.Бцент},
    {"бцент", ТОК.Бцент},
    {"uint", ТОК.Бцел},
    {"бцел", ТОК.Бцел},
    {"ulong", ТОК.Бдол},
    {"бдол", ТОК.Бдол},
    {"union", ТОК.Союз},
    {"союз", ТОК.Союз},
    {"unittest", ТОК.Юниттест},
    {"ushort", ТОК.Бкрат},
    {"бкрат", ТОК.Бкрат},
    {"version", ТОК.Версия},
    {"версия", ТОК.Версия},
    {"void", ТОК.Проц},
    {"проц", ТОК.Проц},
    {"volatile", ТОК.Волатайл},
    {"wchar", ТОК.Шим},
    {"шим", ТОК.Шим},
    {"while", ТОК.Пока},
    {"пока", ТОК.Пока},
    {"with", ТОК.Для},
    {"для", ТОК.Для},
    // Special семы:
    {"__FILE__", ТОК.ФАЙЛ},
    {"__ФАЙЛ__", ТОК.ФАЙЛ},
    {"__LINE__", ТОК.СТРОКА},
    {"__СТРОКА__", ТОК.СТРОКА},
    {"__DATE__", ТОК.ДАТА},
    {"__ДАТА__", ТОК.ДАТА},
    {"__TIME__", ТОК.ВРЕМЯ},
    {"__ВРЕМЯ__", ТОК.ВРЕМЯ},
    {"__TIMESTAMP__", ТОК.ШТАМПВРЕМЕНИ},
    {"__ШТАМПВРЕМЕНИ__", ТОК.ШТАМПВРЕМЕНИ},
    {"__VENDOR__", ТОК.ПОСТАВЩИК},
    {"__ПОСТАВЩИК__", ТОК.ПОСТАВЩИК},
    {"__VERSION__", ТОК.ВЕРСИЯ},
    {"__ВЕРСИЯ__", ТОК.ВЕРСИЯ},
    {"__EOF__", ТОК.КФ}, // D2.0
    {"__КФ__", ТОК.КФ},
];
```

```
module drc.lexer.Lexer;

import drc.lexer.Token,
       drc.lexer.Keywords,
       drc.lexer.Identifier,
       drc.lexer.IdTable;
import drc.Diagnostics;
import drc.Messages;
import drc.HtmlEntities;
import drc.CompilerInfo;
import drc.Unicode;
import drc.SourceText;
import drc.Time;
import common;

import cidrus : strtof, strtod, strtold;
import cidrus : ERANGE;
import core.Vararg;

extern  (C) int getErrno();      // for internal use
extern  (C) int setErrno(int);   // for internal use

alias getErrno errno;
alias setErrno errno;

public import drc.lexer.Funcs;

/// Лексер анализирует символы исходного текста и
/// производит дважды-линкованный список сем (токенов).
class Лексер
{
  ИсходныйТекст исхТекст; /// Исходный текст.
  сим* p;             /// Указывает на текущий символ в исходном тексте.
  сим* конец;           /// Указывает на символ после конца исходного текста.

  Сема* глава;  /// Глава дважды линкованного списка сем.
  Сема* хвост;  /// Хвост линкованного список. Set in сканируй().
  Сема* сема; /// Указывает на текущую сему в списке сем.

  // Members used for ошибка сообщения:
  Диагностика диаг;
  ОшибкаЛексера[] ошибки;
  /// Всегда указывает на первый символ текущей строки.
  сим* началоСтроки;
//   Сема* нс;     /// Current нс сема.
  бцел номСтр = 1;   /// Current, actual source текст line число.
  бцел lineNum_hline; /// Line число установи by #line.
  бцел inTokenString; /// > 0 if внутри q{ }
  /// Holds the original file путь and the modified one (by #line.)
  ДанныеНовСтр.ФПути* путиКФайлам;

  /// Конструировать Лексер объект.
  /// Параметры:
  ///   исхТекст = the UTF-8 source код.
  ///   диаг = used for collecting ошибка сообщения.
  this(ИсходныйТекст исхТекст, Диагностика диаг = null)
  {
    this.исхТекст = исхТекст;
    this.диаг = диаг;

    assert(текст.length && текст[$-1] == 0, "в исходнике отсутствует символ
sentinel");
    this.p = текст.ptr;
```

```
    this.конец = this.р + текст.length;
    this.началоСтроки = this.р;

    this.глава = new Сема;
    this.глава.вид = ТОК.ГОЛОВА;
    this.глава.старт = this.глава.конец = this.р;
    this.сема = this.глава;
    // Initialize this.путиКФайлам.
    новыйПутьФ(this.исхТекст.путьКФайлу);
    // Add a нс as the first сема after the глава.
    auto нс = new Сема;
    нс.вид = ТОК.Новстр;
    нс.установиФлагПробельные();
    нс.старт = нс.конец = this.р;
    нс.нс.путиКФайлам = this.путиКФайлам;
    нс.нс.oriLineNum = 1;
    нс.нс.setLineNum = 0;
    // Link in.
    this.сема.следщ = нс;
    нс.предш = this.сема;
    this.сема = нс;
//      this.нс = нс;
    сканируйШебанг();
  }

  /// The destructor deletes the doubly-linked сема список.
  ~this()
  {
    auto сема = глава.следщ;
    while (сема !is null)
    {
      assert(сема.вид == ТОК.КФ ? сема == хвост && сема.следщ is null : 1);
      delete сема.предш;
      сема = сема.следщ;
    }
    delete хвост;
  }

  ткст текст()
  {
    return исхТекст.данные;
  }

  /// The "shebang" may optionally appear once at the beginning of a file.
  /// Regexp: #![^\EndOfLine]*
  проц  сканируйШебанг()
  {
    if (*р == '#' && р[1] == '!')
    {
      auto т = new Сема;
      т.вид = ТОК.Шебанг;
      т.установиФлагПробельные();
      т.старт = р;
      ++р;
      while (!конецСтроки_ли(++р))
        аски_ли(*р) || раскодируйЮ8();
      т.конец = р;
      this.сема.следщ = т;
      т.предш = this.сема;
    }
  }

  /// Sets the значение of the special сема.
```

```
  проц  закончиОсобуюСему(ref Сема т)
  {
    assert(т.исхТекст[0..2] == "__");
    switch (т.вид)
    {
    case ТОК.ФАЙЛ:
      т.ткт = this.путиКФайлам.устПуть;
      break;
    case ТОК.СТРОКА:
      т.бцел_ = this.номерСтрокиОшиб(this.номСтр);
      break;
    case ТОК.ДАТА,
         ТОК.ВРЕМЯ,
         ТОК.ШТАМПВРЕМЕНИ:
      auto ткт_время = Время.вТкст();
      switch (т.вид)
      {
      case ТОК.ДАТА:
        ткт_время = Время.день_месяца(ткт_время) ~ ' ' ~
Время.год(ткт_время); break;
      case ТОК.ВРЕМЯ:
        ткт_время = Время.время(ткт_время); break;
      case ТОК.ШТАМПВРЕМЕНИ:
        break; // ткт_время is the timestamp.
      default: assert(0);
      }
      ткт_время ~= '\0'; // Terminate with a zero.
      т.ткт = ткт_время;
      break;
    case ТОК.ПОСТАВЩИК:
      т.ткт = ПОСТАВЩИК;
      break;
    case ТОК.ВЕРСИЯ:
      т.бцел_ = VERSION_MAJOR*1000 + VERSION_MINOR;
      break;
    default:
      assert(0);
    }
  }

  /// Sets a new file путь.
  проц  новыйПутьФ(ткст новПуть)
  {
    auto пути = new ДанныеНовСтр.ФПути;
    пути.исхПуть = this.исхТекст.путьКФайлу;
    пути.устПуть = новПуть;
    this.путиКФайлам = пути;
  }

  private проц  установиНачалоСтроки(сим* р)
  {
    // Check that we can look behind one символ.
    assert((р-1) >= текст.ptr && р < конец);
    // Check that предшious символ is a нс.
    assert(конецНовСтроки_ли(р - 1));
    this.началоСтроки = р;
  }

  /// Scans the следщ сема in the source текст.
  ///
  /// Creates a new сема if т.следщ is null and appends it в the список.
  private проц  сканируйСледщ(ref Сема* т)
  {
```

```
    assert(t !is null);
    if (t.следщ)
    {
      t = t.следщ;
//       if (t.вид == ТОК.Новстр)
//         this.нс = t;
    }
    else if (t != this.хвост)
    {
      Сема* т_нов = new Сема;
      сканируй(*т_нов);
      т_нов.предш = t;
      t.следщ = т_нов;
      t = т_нов;
    }
  }

  /// Advance t one сема forward.
  проц  возьми(ref Сема* t)
  {
    сканируйСледщ(t);
  }

  /// Advance в the следщ сема in the source текст.
  ТОК следщСема()
  {
    сканируйСледщ(this.сема);
    return this.сема.вид;
  }

  /// Returns да if p points в the last символ of a Новстр.
  бул конецНовСтроки_ли(сим* p)
  {
    if (*p == '\n' || *p == '\r')
      return да;
    if (*p == PC[2] || *p == PA[2])
      if ((p-2) >= текст.ptr)
        if (p[-1] == PC[1] && p[-2] == PC[0])
          return да;
    return нет;
  }

  /// The main method which recognizes the characters that make up a сема.
  ///
  /// Complicated семы are scanned in separate methods.
  public проц  сканируй(ref Сема t)
  in
  {
    assert(текст.ptr <= p && p < конец);
  }
  out
  {
    assert(текст.ptr <= t.старт && t.старт < конец, Сема.вТкст(t.вид));
    assert(текст.ptr <= t.конец && t.конец <= конец, Сема.вТкст(t.вид));
  }
  body
  {
    // Scan whitespace.
    if (пбел_ли(*p))
    {
      t.пп = p;
      while (пбел_ли(*++p))
      {}
```

```
    }

    // Scan a сема.
    бцел с = *р;
    {
      т.старт = р;
      // Новстр.
      switch (*р)
      {
      case '\r':
        if (р[1] == '\n')
          ++р;
      case '\n':
        assert(конецНовСтроки_ли(р));
        ++р;
        ++номСтр;
        установиНачалоСтроки(р);
//          this.нс = &т;
        т.вид = ТОК.Новстр;
        т.установиФлагПробельные();
        т.нс.путиКФайлам = this.путиКФайлам;
        т.нс.oriLineNum = номСтр;
        т.нс.setLineNum = lineNum_hline;
        т.конец = р;
        return;
      default:
        if (новСтрЮ_ли(р))
        {
          ++р; ++р;
          goto case '\n';
        }
      }
      // Идентификатор or ткст literal.
      if (начсим_ли(с))
      {
        if (с == 'r' && р[1] == '"' && ++р)
          return scanRawStringLiteral(т);
        if (с == 'x' && р[1] == '"')
          return scanHexStringLiteral(т);
        version(D2)
        {
        if (с == 'q' && р[1] == '"')
          return scanDelimitedStringLiteral(т);
        if (с == 'q' && р[1] == '{')
          return scanTokenStringLiteral(т);
        }
        // Scan identifier.
      Lidentifier:
        do
        { с = *++р; }
        while (идент_ли(с) || !аски_ли(с) && юАльфа_ли())

        т.конец = р;

        auto ид = ТаблицаИд.сыщи(т.исхТекст);
        т.вид = ид.вид;
        т.идент = ид;

        if (т.вид == ТОК.Идентификатор || т.кслово_ли)
          return;
        else if (т.спецСема_ли)
          закончиОсобуюСему(т);
        else if (т.вид == ТОК.КФ)
```

```
    {
      хвост = &t;
      assert(t.исхТекст == "__EOF__");
    }
    else
      assert(0, "неожидаемый тип семы: " ~ Сема.вТкст(t.вид));
    return;
  }

  if (цифра_ли(c))
    return scanNumber(t);

  if (c == '/')
  {
    c = *++p;
    switch(c)
    {
    case '=':
      ++p;
      t.вид = ТОК.ДелениеПрисвой;
      t.конец = p;
      return;
    case '+':
      return scanNestedComment(t);
    case '*':
      return scanBlockComment(t);
    case '/':
      while (!конецСтроки_ли(++p))
        аски_ли(*p) || раскодируйЮ8();
      t.вид = ТОК.Комментарий;
      t.установиФлагПробельные();
      t.конец = p;
      return;
    default:
      t.вид = ТОК.Деление;
      t.конец = p;
      return;
    }
  }

  switch (c)
  {
  case '\'':
    return сканируйСимвольныйЛитерал(t);
  case '`':
    return scanRawStringLiteral(t);
  case '"':
    return scanNormalStringLiteral(t);
  case '\\':
    ткст буфер;
    do
    {
      бул isBinary;
      c = scanEscapeSequence(isBinary);
      if (аски_ли(c) || isBinary)
        буфер ~= c;
      else
        encodeUTF8(буфер, c);
    } while (*p == '\\')
    буфер ~= 0;
    t.вид = ТОК.Ткст;
    t.ткт = буфер;
    t.конец = p;
```

```
    return;
  case '>': /* >  >=  >>  >>=  >>>  >>>= */
    c = *++p;
    switch (c)
    {
    case '=':
      t.вид = ТОК.БольшеРавно;
      goto Lcommon;
    case '>':
      if (p[1] == '>')
      {
        ++p;
        if (p[1] == '=')
        { ++p;
          t.вид = ТОК.URShiftAssign;
        }
        else
          t.вид = ТОК.URShift;
      }
      else if (p[1] == '=')
      {
        ++p;
        t.вид = ТОК.ПСдвигПрисвой;
      }
      else
        t.вид = ТОК.ПСдвиг;
      goto Lcommon;
    default:
      t.вид = ТОК.Больше;
      goto Lcommon2;
    }
    assert(0);
  case '<': /* <  <=  <>  <>=  <<  <<= */
    c = *++p;
    switch (c)
    {
    case '=':
      t.вид = ТОК.МеньшеРавно;
      goto Lcommon;
    case '<':
      if (p[1] == '=') {
        ++p;
        t.вид = ТОК.ЛСдвигПрисвой;
      }
      else
        t.вид = ТОК.ЛСдвиг;
      goto Lcommon;
    case '>':
      if (p[1] == '=') {
        ++p;
        t.вид = ТОК.LorEorG;
      }
      else
        t.вид = ТОК.LorG;
      goto Lcommon;
    default:
      t.вид = ТОК.Меньше;
      goto Lcommon2;
    }
    assert(0);
  case '!': /* !  !<  !>  !<=  !>=  !<>  !<>= */
    c = *++p;
    switch (c)
```

```
    {
    case '<':
      c = *++p;
      if (c == '>')
      {
        if (p[1] == '=') {
          ++p;
          t.вид = ТОК.Unordered;
        }
        else
          t.вид = ТОК.UorE;
      }
      else if (c == '=')
      {
        t.вид = ТОК.UorG;
      }
      else {
        t.вид = ТОК.UorGorE;
        goto Lcommon2;
      }
      goto Lcommon;
    case '>':
      if (p[1] == '=')
      {
        ++p;
        t.вид = ТОК.UorL;
      }
      else
        t.вид = ТОК.UorLorE;
      goto Lcommon;
    case '=':
      t.вид = ТОК.НеРавно;
      goto Lcommon;
    default:
      t.вид = ТОК.Не;
      goto Lcommon2;
    }
    assert(0);
  case '.': /* .  .[0-9]  ..  ... */
    if (p[1] == '.')
    {
      ++p;
      if (p[1] == '.') {
        ++p;
        t.вид = ТОК.Эллипсис;
      }
      else
        t.вид = ТОК.Срез;
    }
    else if (цифра_ли(p[1]))
    {
      return scanReal(t);
    }
    else
      t.вид = ТОК.Точка;
    goto Lcommon;
  case '|': /* |  ||  |= */
    c = *++p;
    if (c == '=')
      t.вид = ТОК.ИлиПрисвой;
    else if (c == '|')
      t.вид = ТОК.ИлиЛогическое;
    else {
```

```
      t.вид = ТОК.ИлиБинарное;
      goto Lcommon2;
    }
    goto Lcommon;
  case '&': /* &  &&  &= */
    c = *++p;
    if (c == '=')
      t.вид = ТОК.ИПрисвой;
    else if (c == '&')
      t.вид = ТОК.ИЛогическое;
    else {
      t.вид = ТОК.ИБинарное;
      goto Lcommon2;
    }
    goto Lcommon;
  case '+': /* +  ++  += */
    c = *++p;
    if (c == '=')
      t.вид = ТОК.ПлюсПрисвой;
    else if (c == '+')
      t.вид = ТОК.ПлюсПлюс;
    else {
      t.вид = ТОК.Плюс;
      goto Lcommon2;
    }
    goto Lcommon;
  case '-': /* -  --  -= */
    c = *++p;
    if (c == '=')
      t.вид = ТОК.МинусПрисвой;
    else if (c == '-')
      t.вид = ТОК.МинусМинус;
    else {
      t.вид = ТОК.Минус;
      goto Lcommon2;
    }
    goto Lcommon;
  case '=': /* =  == */
    if (p[1] == '=') {
      ++p;
      t.вид = ТОК.Равно;
    }
    else
      t.вид = ТОК.Присвоить;
    goto Lcommon;
  case '~': /* ~  ~= */
     if (p[1] == '=') {
       ++p;
       t.вид = ТОК.CatAssign;
     }
     else
       t.вид = ТОК.Тильда;
     goto Lcommon;
  case '*': /* *  *= */
     if (p[1] == '=') {
       ++p;
       t.вид = ТОК.УмножьПрисвой;
     }
     else
       t.вид = ТОК.Умножь;
     goto Lcommon;
  case '^': /* ^  ^= */
     if (p[1] == '=') {
```

```
      ++p;
      t.вид = ТОК.ИИлиПрисвой;
    }
    else
      t.вид = ТОК.ИИли;
    goto Lcommon;
case '%': /* %  %= */
    if (p[1] == '=') {
      ++p;
      t.вид = ТОК.МодульПрисвой;
    }
    else
      t.вид = ТОК.Модуль;
    goto Lcommon;
// Single символ семы:
case '(':
  t.вид = ТОК.ЛСкобка;
  goto Lcommon;
case ')':
  t.вид = ТОК.ПСкобка;
  goto Lcommon;
case '[':
  t.вид = ТОК.ЛКвСкобка;
  goto Lcommon;
case ']':
  t.вид = ТОК.ПКвСкобка;
  goto Lcommon;
case '{':
  t.вид = ТОК.ЛФСкобка;
  goto Lcommon;
case '}':
  t.вид = ТОК.ПФСкобка;
  goto Lcommon;
case ':':
  t.вид = ТОК.Двоеточие;
  goto Lcommon;
case ';':
  t.вид = ТОК.ТочкаЗапятая;
  goto Lcommon;
case '?':
  t.вид = ТОК.Вопрос;
  goto Lcommon;
case ',':
  t.вид = ТОК.Запятая;
  goto Lcommon;
case '$':
  t.вид = ТОК.Доллар;
Lcommon:
  ++p;
Lcommon2:
  t.конец = p;
  return;
case '#':
  return scanSpecialTokenSequence(t);
default:
}

// Check for КФ
if (кф_ли(c))
{
  assert(кф_ли(*p), ""~*p);
  t.вид = ТОК.КФ;
  t.конец = p;
```

```
      хвост = &t;
      assert(t.старт == t.конец);
      return;
    }

    if (!аски_ли(с))
    {
      с = раскодируйЮ8();
      if (униАльфа_ли(с))
        goto Lidentifier;
    }

    ошибка(t.старт, ИДС.НедопустимыйСимвол, cast(дим)с);

    ++р;
    t.вид = ТОК.Нелегал;
    t.установиФлагПробельные();
    t.дим_ = с;
    t.конец = р;
    return;
  }
}

/// Converts a ткст literal в an integer.
template тоБцел(ткст Т)
{
  static assert(0 < Т.length && Т.length <= 4);
  static if (Т.length == 1)
    const бцел тоБцел = Т[0];
  else
    const бцел тоБцел = (Т[0] << ((Т.length-1)*8)) | тоБцел!(Т[1..$]);
}
static assert(тоБцел!("\xAA\xBB\xCC\xDD") == 0xAABBCCDD);

/// Constructs case инструкции. E.g.:
/// ---
//// // case_!("<", "Меньше", "Lcommon") ->
/// case 60u:
///   t.вид = ТОК.Меньше;
///   goto Lcommon;
/// ---
/// FIXME: Can't use this yet due в a $(DMDBUG 1534, bug) in DMD.
template case_(ткст ткт, ткст вид, ткст лейбл)
{
  const ткст case_ =
    `case `~тоБцел!(ткт).stringof~`:`
      `t.вид = ТОК.`~вид~`;`
      `goto `~лейбл~`;`;
}
//pragma(сооб, case_!("<", "Меньше", "Lcommon"));

template case_L4(ткст ткт, ТОК вид)
{
  const ткст case_L4 = case_!(ткт, вид, "Lcommon_4");
}

template case_L3(ткст ткт, ТОК вид)
{
  const ткст case_L3 = case_!(ткт, вид, "Lcommon_3");
}

template case_L2(ткст ткт, ТОК вид)
{
```

```
    const ткст case_L2 = case_!(ткт, вид, "Lcommon_2");
  }

  template case_L1(ткст ткт, ТОК вид)
  {
    const ткст case_L3 = case_!(ткт, вид, "Lcommon");
  }

  /// An alternative сканируй method.
  /// Profiling shows it's a bit slower.
  public проц  scan_(ref Сема t)
  in
  {
    assert(текст.ptr <= p && p < конец);
  }
  out
  {
    assert(текст.ptr <= t.старт && t.старт < конец, Сема.вТкст(t.вид));
    assert(текст.ptr <= t.конец && t.конец <= конец, Сема.вТкст(t.вид));
  }
  body
  {
    // Scan whitespace.
    if (пбел_ли(*p))
    {
      t.пп = p;
      while (пбел_ли(*++p))
      {}
    }

    // Scan a сема.
    t.старт = p;
    // Новстр.
    switch (*p)
    {
    case '\r':
      if (p[1] == '\n')
        ++p;
    case '\n':
      assert(конецНовСтроки_ли(p));
      ++p;
      ++номСтр;
      установиНачалоСтроки(p);
//       this.нс = &t;
      t.вид = ТОК.Новстр;
      t.установиФлагПробельные();
      t.нс.путиКФайлам = this.путиКФайлам;
      t.нс.oriLineNum = номСтр;
      t.нс.setLineNum = lineNum_hline;
      t.конец = p;
      return;
    default:
      if (новСтрЮ_ли(p))
      {
        ++p; ++p;
        goto case '\n';
      }
    }

    бцел c = *p;
    assert(конец - p != 0);
    switch (конец - p)
    {
```

```
    case 1:
      goto L1character;
    case 2:
      c <<= 8; c |= p[1];
      goto L2characters;
    case 3:
      c <<= 8; c |= p[1]; c <<= 8; c |= p[2];
      goto L3characters;
    default:
      version(BigEndian)
        c = *cast(бцел*)p;
      else
      {
        c <<= 8; c |= p[1]; c <<= 8; c |= p[2]; c <<= 8; c |= p[3];
        /+
        c = *cast(бцел*)p;
        asm
        {
          mov EDX, c;
          bswap EDX;
          mov c, EDX;
        }
        +/
      }
    }

    // 4 символ семы.
    switch (c)
    {
    case тоБцел!(">>>="):
      т.вид = ТОК.ПСдвигПрисвой;
      goto Lcommon_4;
    case тоБцел!("!<>="):
      т.вид = ТОК.Unordered;
    Lcommon_4:
      p += 4;
      т.конец = p;
      return;
    default:
    }

    c >>>= 8;
  L3characters:
    assert(p == т.старт);
    // 3 символ семы.
    switch (c)
    {
    case тоБцел!(">>="):
      т.вид = ТОК.ПСдвигПрисвой;
      goto Lcommon_3;
    case тоБцел!(">>>"):
      т.вид = ТОК.URShift;
      goto Lcommon_3;
    case тоБцел!("<>="):
      т.вид = ТОК.LorEorG;
      goto Lcommon_3;
    case тоБцел!("<<="):
      т.вид = ТОК.ЛСдвигПрисвой;
      goto Lcommon_3;
    case тоБцел!("!<="):
      т.вид = ТОК.UorG;
      goto Lcommon_3;
    case тоБцел!("!>="):
```

```
      t.вид = ТОК.UorL;
      goto Lcommon_3;
    case тоБцел!("!<>"):
      t.вид = ТОК.UorE;
      goto Lcommon_3;
    case тоБцел!("..."):
      t.вид = ТОК.Эллипсис;
    Lcommon_3:
      p += 3;
      t.конец = p;
      return;
    default:
    }

    c >>>= 8;
  L2characters:
    assert(p == t.старт);
    // 2 символ семы.
    switch (c)
    {
    case тоБцел!("/+"):
      ++p; // Skip /
      return scanNestedComment(t);
    case тоБцел!("/*"):
      ++p; // Skip /
      return scanBlockComment(t);
    case тоБцел!("//"):
      ++p; // Skip /
      assert(*p == '/');
      while (!конецСтроки_ли(++p))
        аски_ли(*p) || раскодируйЮ8();
      t.вид = ТОК.Комментарий;
      t.установиФлагПробельные();
      t.конец = p;
      return;
    case тоБцел!(">="):
      t.вид = ТОК.БольшеРавно;
      goto Lcommon_2;
    case тоБцел!(">>"):
      t.вид = ТОК.ПСдвиг;
      goto Lcommon_2;
    case тоБцел!("<<"):
      t.вид = ТОК.ЛСдвиг;
      goto Lcommon_2;
    case тоБцел!("<="):
      t.вид = ТОК.МеньшеРавно;
      goto Lcommon_2;
    case тоБцел!("<>"):
      t.вид = ТОК.LorG;
      goto Lcommon_2;
    case тоБцел!("!<"):
      t.вид = ТОК.UorGorE;
      goto Lcommon_2;
    case тоБцел!("!>"):
      t.вид = ТОК.UorLorE;
      goto Lcommon_2;
    case тоБцел!("!="):
      t.вид = ТОК.НеРавно;
      goto Lcommon_2;
    case тоБцел!(".."):
      t.вид = ТОК.Срез;
      goto Lcommon_2;
    case тоБцел!("&&"):
```

```
      t.вид = ТОК.ИЛогическое;
      goto Lcommon_2;
    case тоБцел!("&="):
      t.вид = ТОК.ИПрисвой;
      goto Lcommon_2;
    case тоБцел!("||"):
      t.вид = ТОК.ИлиЛогическое;
      goto Lcommon_2;
    case тоБцел!("|="):
      t.вид = ТОК.ИлиПрисвой;
      goto Lcommon_2;
    case тоБцел!("++"):
      t.вид = ТОК.ПлюсПлюс;
      goto Lcommon_2;
    case тоБцел!("+="):
      t.вид = ТОК.ПлюсПрисвой;
      goto Lcommon_2;
    case тоБцел!("--"):
      t.вид = ТОК.МинусМинус;
      goto Lcommon_2;
    case тоБцел!("-="):
      t.вид = ТОК.МинусПрисвой;
      goto Lcommon_2;
    case тоБцел!("=="):
      t.вид = ТОК.Равно;
      goto Lcommon_2;
    case тоБцел!("~="):
      t.вид = ТОК.CatAssign;
      goto Lcommon_2;
    case тоБцел!("*="):
      t.вид = ТОК.УмножьПрисвой;
      goto Lcommon_2;
    case тоБцел!("/="):
      t.вид = ТОК.ДелениеПрисвой;
      goto Lcommon_2;
    case тоБцел!("^="):
      t.вид = ТОК.ИИлиПрисвой;
      goto Lcommon_2;
    case тоБцел!("%="):
      t.вид = ТОК.МодульПрисвой;
    Lcommon_2:
      p += 2;
      t.конец = p;
      return;
    default:
    }

    c >>>= 8;
  L1character:
    assert(p == t.старт);
    assert(*p == c, Формат("p={0},c={1}", *p, cast(дим)c));
    // 1 символ семы.
    // TODO: consторонar storing the сема тип in ptable.
    switch (c)
    {
    case '\'':
      return сканируйСимвольныйЛитерал(t);
    case '`':
      return scanRawStringLiteral(t);
    case '"':
      return scanNormalStringLiteral(t);
    case '\\':
      ткст буфер;
```

```
  do
  {
    бул isBinary;
    c = scanEscapeSequence(isBinary);
    if (аски_ли(c) || isBinary)
      буфер ~= c;
    else
      encodeUTF8(буфер, c);
  } while (*p == '\\')
  буфер ~= 0;
  t.вид = ТОК.Ткст;
  t.ткт = буфер;
  t.конец = p;
  return;
case '<':
  t.вид = ТОК.Больше;
  goto Lcommon;
case '>':
  t.вид = ТОК.Меньше;
  goto Lcommon;
case '^':
  t.вид = ТОК.ИИли;
  goto Lcommon;
case '!':
  t.вид = ТОК.Не;
  goto Lcommon;
case '.':
  if (цифра_ли(p[1]))
    return scanReal(t);
  t.вид = ТОК.Точка;
  goto Lcommon;
case '&':
  t.вид = ТОК.ИБинарное;
  goto Lcommon;
case '|':
  t.вид = ТОК.ИлиБинарное;
  goto Lcommon;
case '+':
  t.вид = ТОК.Плюс;
  goto Lcommon;
case '-':
  t.вид = ТОК.Минус;
  goto Lcommon;
case '=':
  t.вид = ТОК.Присвоить;
  goto Lcommon;
case '~':
  t.вид = ТОК.Тильда;
  goto Lcommon;
case '*':
  t.вид = ТОК.Умножь;
  goto Lcommon;
case '/':
  t.вид = ТОК.Деление;
  goto Lcommon;
case '%':
  t.вид = ТОК.Модуль;
  goto Lcommon;
case '(':
  t.вид = ТОК.ЛСкобка;
  goto Lcommon;
case ')':
  t.вид = ТОК.ПСкобка;
```

```
  goto Lcommon;
case '[':
  t.вид = ТОК.ЛКвСкобка;
  goto Lcommon;
case ']':
  t.вид = ТОК.ПКвСкобка;
  goto Lcommon;
case '{':
  t.вид = ТОК.ЛФСкобка;
  goto Lcommon;
case '}':
  t.вид = ТОК.ПФСкобка;
  goto Lcommon;
case ':':
  t.вид = ТОК.Двоеточие;
  goto Lcommon;
case ';':
  t.вид = ТОК.ТочкаЗапятая;
  goto Lcommon;
case '?':
  t.вид = ТОК.Вопрос;
  goto Lcommon;
case ',':
  t.вид = ТОК.Запятая;
  goto Lcommon;
case '$':
  t.вид = ТОК.Доллар;
Lcommon:
  ++p;
  t.конец = p;
  return;
case '#':
  return scanSpecialTokenSequence(t);
default:
}

assert(p == t.старт);
assert(*p == c);

// TODO: сопсторонаr moving начсим_ли() and цифра_ли() up.
if (начсим_ли(c))
{
  if (c == 'r' && p[1] == '"' && ++p)
    return scanRawStringLiteral(t);
  if (c == 'x' && p[1] == '"')
    return scanHexStringLiteral(t);
version(D2)
{
  if (c == 'q' && p[1] == '"')
    return scanDelimitedStringLiteral(t);
  if (c == 'q' && p[1] == '{')
    return scanTokenStringLiteral(t);
}
  // Scan identifier.
Lidentifier:
  do
  { c = *++p; }
  while (идент_ли(c) || !аски_ли(c) && юАльфа_ли())

  t.конец = p;

  auto ид = ТаблицаИд.сыщи(t.исхТекст);
  t.вид = ид.вид;
```

```
    t.идент = ид;

    if (t.вид == ТОК.Идентификатор || t.кслово_ли)
      return;
    else if (t.спецСема_ли)
      закончиОсобуюСему(t);
    else if (t.вид == ТОК.КФ)
    {
      хвост = &t;
      assert(t.исхТекст == "__EOF__");
    }
    else
      assert(0, "unexpected сема тип: " ~ Сема.вТкст(t.вид));
    return;
  }

  if (цифра_ли(c))
    return scanNumber(t);

  // Check for КФ
  if (кф_ли(c))
  {
    assert(кф_ли(*p), *p~"");
    t.вид = ТОК.КФ;
    t.конец = p;
    хвост = &t;
    assert(t.старт == t.конец);
    return;
  }

  if (!аски_ли(c))
  {
    c = раскодируйЮ8();
    if (униАльфа_ли(c))
      goto Lidentifier;
  }

  ошибка(t.старт, ИДС.НедопустимыйСимвол, cast(дим)c);

  ++p;
  t.вид = ТОК.Нелегал;
  t.установиФлагПробельные();
  t.дим_ = c;
  t.конец = p;
  return;
}

/// Scans a block comment.
///
/// BlockComment := "/*" AnyChar* "*/"
проц  scanBlockComment(ref Сема t)
{
  assert(p[-1] == '/' && *p == '*');
  auto tokenLineNum = номСтр;
  auto tokenLineBegin = началоСтроки;
Loop:
  while (1)
  {
    switch (*++p)
    {
    case '*':
      if (p[1] != '/')
        continue;
```

```
        p += 2;
        break Loop;
      case '\r':
        if (p[1] == '\n')
          ++p;
      case '\n':
        assert(конецНовСтроки_ли(p));
        ++номСтр;
        установиНачалоСтроки(p+1);
        break;
      default:
        if (!аски_ли(*p))
        {
          if (симНовСтрЮ_ли(раскодируйЮ8()))
            goto case '\n';
        }
        else if (кф_ли(*p))
        {
          ошибка(tokenLineNum, tokenLineBegin, t.старт,
ИДС.UnterminatedBlockComment);
          break Loop;
        }
      }
    }
    t.вид = ТОК.Комментарий;
    t.установиФлагПробельные();
    t.конец = p;
    return;
  }

  /// Scans a nested comment.
  ///
  /// NestedComment := "/+" (AnyChar* | NestedComment) "+/"
  проц  scanNestedComment(ref Сема t)
  {
    assert(p[-1] == '/' && *p == '+');
    auto tokenLineNum = номСтр;
    auto tokenLineBegin = началоСтроки;
    бцел уровень = 1;
  Loop:
    while (1)
    {
      switch (*++p)
      {
      case '/':
        if (p[1] == '+')
          ++p, ++уровень;
        continue;
      case '+':
        if (p[1] != '/')
          continue;
        ++p;
        if (--уровень != 0)
          continue;
        ++p;
        break Loop;
      case '\r':
        if (p[1] == '\n')
          ++p;
      case '\n':
        assert(конецНовСтроки_ли(p));
        ++номСтр;
        установиНачалоСтроки(p+1);
```

```
        continue;
      default:
        if (!аски_ли(*р))
        {
          if (симНовСтрЮ_ли(раскодируйЮ8()))
            goto case '\n';
        }
        else if (кф_ли(*р))
        {
          ошибка(tokenLineNum, tokenLineBegin, т.старт,
ИДС.UnterminatedNestedComment);
          break Loop;
        }
      }
    }
    т.вид = ТОК.Комментарий;
    т.установиФлагПробельные();
    т.конец = р;
    return;
  }

  /// Scans the postfix символ of a ткст literal.
  ///
  /// PostfixChar := "c" | "w" | "d"
  сим scanPostfix()
  {
    assert(р[-1] == '"' || р[-1] == '`' ||
      { version(D2) return р[-1] == '}';
              else return 0; }()
    );
    switch (*р)
    {
    case 'c':
    case 'w':
    case 'd':
      return *р++;
    default:
      return 0;
    }
    assert(0);
  }

  /// Scans a normal ткст literal.
  ///
  /// NormalStringLiteral := "\"" Сим* "\""
  проц  scanNormalStringLiteral(ref Сема т)
  {
    assert(*р == '"');
    auto tokenLineNum = номСтр;
    auto tokenLineBegin = началоСтроки;
    т.вид = ТОК.Ткст;
    ткст буфер;
    бцел с;
    while (1)
    {
      с = *++р;
      switch (с)
      {
      case '"':
        ++р;
        т.pf = scanPostfix();
      Lreturn:
        т.ткт = буфер ~ '\0';
```

```
      t.конец = p;
      return;
    case '\\':
      бул isBinary;
      c = scanEscapeSequence(isBinary);
      --p;
      if (аски_ли(c) || isBinary)
        буфер ~= c;
      else
        encodeUTF8(буфер, c);
      continue;
    case '\r':
      if (p[1] == '\n')
        ++p;
    case '\n':
      assert(конецНовСтроки_ли(p));
      c = '\n'; // Convert Новстр в \n.
      ++номСтр;
      установиНачалоСтроки(p+1);
      break;
    case 0, _Z_:
      ошибка(tokenLineNum, tokenLineBegin, t.старт, ИДС.НеоконченыйТкст);
      goto Lreturn;
    default:
      if (!аски_ли(c))
      {
        c = раскодируйЮ8();
        if (симНовСтрЮ_ли(c))
          goto case '\n';
        encodeUTF8(буфер, c);
        continue;
      }
    }
    assert(аски_ли(c));
    буфер ~= c;
  }
  assert(0);
}

/// Scans a символ literal.
///
/// СимЛитерал := "'" Сим "'"
проц  сканируйСимвольныйЛитерал(ref Сема t)
{
  assert(*p == '\'');
  ++p;
  t.вид = ТОК.СимЛитерал;
  switch (*p)
  {
  case '\\':
    бул notused;
    t.дим_ = scanEscapeSequence(notused);
    break;
  case '\'':
    ошибка(t.старт, ИДС.ПустойСимвольныйЛитерал);
    break;
  default:
    if (конецСтроки_ли(p))
      break;
    бцел c = *p;
    if (!аски_ли(c))
      c = раскодируйЮ8();
    t.дим_ = c;
```

```
      ++p;
    }

    if (*p == '\'')
      ++p;
    else
      ошибка(t.старт, ИДС.НеоконченныйСимвольныйЛитерал);
    t.конец = p;
  }

  /// Scans a raw ткст literal.
  ///
  /// RawStringLiteral := "r\"" AnyChar* "\"" | "`" AnyChar* "`"
  проц  scanRawStringLiteral(ref Сема t)
  {
    assert(*p == '`' || *p == '"' && p[-1] == 'r');
    auto tokenLineNum = номСтр;
    auto tokenLineBegin = началоСтроки;
    t.вид = ТОК.Ткст;
    бцел delim = *p;
    ткст буфер;
    бцел c;
    while (1)
    {
      c = *++p;
      switch (c)
      {
      case '\r':
        if (p[1] == '\n')
          ++p;
      case '\n':
        assert(конецНовСтроки_ли(p));
        c = '\n'; // Convert Новстр в '\n'.
        ++номСтр;
        установиНачалоСтроки(p+1);
        break;
      case '`':
      case '"':
        if (c == delim)
        {
          ++p;
          t.pf = scanPostfix();
        Lreturn:
          t.ткт = буфер ~ '\0';
          t.конец = p;
          return;
        }
        break;
      case 0, _Z_:
        ошибка(tokenLineNum, tokenLineBegin, t.старт,
          delim == 'r' ? ИДС.UnterminatedRawString :
ИДС.UnterminatedBackQuoteString);
        goto Lreturn;
      default:
        if (!аски_ли(c))
        {
          c = раскодируйЮ8();
          if (симНовСтрЮ_ли(c))
            goto case '\n';
          encodeUTF8(буфер, c);
          continue;
        }
      }
```

```
      assert(аски_ли(c));
      буфер ~= c;
    }
    assert(0);
  }

  /// Scans a hexadecimal ткст literal.
  ///
  /// HexStringLiteral := "x\"" (HexChar HexChar)* "\""
  проц  scanHexStringLiteral(ref Сема t)
  {
    assert(p[0] == 'x' && p[1] == '"');
    t.вид = ТОК.Ткст;

    auto tokenLineNum = номСтр;
    auto tokenLineBegin = началоСтроки;

    бцел c;
    ббайт[] буфер;
    ббайт h; // hex число
    бцел n; // число of hex digits

    ++p;
    assert(*p == '"');
    while (1)
    {
      c = *++p;
      switch (c)
      {
      case '"':
        if (n & 1)
          ошибка(tokenLineNum, tokenLineBegin, t.старт,
ИДС.OddNumberOfDigitsInHexString);
        ++p;
        t.pf = scanPostfix();
      Lreturn:
        t.ткт = cast(ткст) (буфер ~= 0);
        t.конец = p;
        return;
      case '\r':
        if (p[1] == '\n')
          ++p;
      case '\n':
        assert(конецНовСтроки_ли(p));
        ++номСтр;
        установиНачалоСтроки(p+1);
        continue;
      default:
        if (гекс_ли(c))
        {
          if (c <= '9')
            c -= '0';
          else if (c <= 'F')
            c -= 'A' - 10;
          else
            c -= 'a' - 10;

          if (n & 1)
          {
            h <<= 4;
            h |= c;
            буфер ~= h;
          }
```

```
          else
            h = cast(ббайт)c;
          ++n;
          continue;
        }
        else if (пбел_ли(c))
          continue; // Skip spaces.
        else if (кф_ли(c))
        {
          ошибка(tokenLineNum, tokenLineBegin, t.старт,
ИДС.UnterminatedHexString);
          t.pf = 0;
          goto Lreturn;
        }
        else
        {
          auto errorAt = p;
          if (!аски_ли(c))
          {
            c = раскодируйЮ8();
            if (симНовСтрЮ_ли(c))
              goto case '\n';
          }
          ошибка(errorAt, ИДС.NonHexCharInHexString, cast(дим)c);
        }
      }
    }
    assert(0);
  }

version(DDoc)
{
  /// Scans a delimited ткст literal.
  проц  scanDelimitedStringLiteral(ref Сема t);
  /// Scans a сема ткст literal.
  ///
  /// TokenStringLiteral := "q{" Сема* "}"
  проц  scanTokenStringLiteral(ref Сема t);
}
else
version(D2)
{
  проц  scanDelimitedStringLiteral(ref Сема t)
  {
    assert(p[0] == 'q' && p[1] == '"');
    t.вид = ТОК.Ткст;

    auto tokenLineNum = номСтр;
    auto tokenLineBegin = началоСтроки;

    ткст буфер;
    дим открывающий_delim = 0, // 0 if no nested delimiter or '[', '(', '<',
'{'
        закрывающий_delim; // Will be ']', ')', '>', '},
                           // the first символ of an identifier or
                           // any другой Unicode/ASCII символ.
    ткст ткт_delim; // Идентификатор delimiter.
    бцел уровень = 1; // Counter for nestable delimiters.

    ++p; ++p; // Skip q"
    бцел c = *p;
    switch (c)
    {
```

```
    case '(':
      открывающий_delim = c;
      закрывающий_delim = ')'; // c + 1
      break;
    case '[', '<', '{':
      открывающий_delim = c;
      закрывающий_delim = c + 2; // Get в закрывающий counterpart. Feature of
ASCII таблица.
      break;
    default:
      дим сканируйНовСтр()
      {
        switch (*p)
        {
        case '\r':
          if (p[1] == '\n')
            ++p;
        case '\n':
          assert(конецНовСтроки_ли(p));
          ++p;
          ++номСтр;
          установиНачалоСтроки(p);
          break;
        default:
          if (новСтрЮ_ли(p)) {
            p += 2;
            goto case '\n';
          }
          return нет;
        }
        return да;
      }
      // Skip leading newlines:
      while (сканируйНовСтр())
      {}
      assert(!новСтр_ли(p));

      сим* начало = p;
      c = *p;
      закрывающий_delim = c;
      // TODO: Check for non-printable characters?
      if (!аски_ли(c))
      {
        закрывающий_delim = раскодируйЮ8();
        if (!униАльфа_ли(закрывающий_delim))
          break; // Не an identifier.
      }
      else if (!начсим_ли(c))
        break; // Не an identifier.

      // Parse Идентификатор + EndOfLine
      do
      { c = *++p; }
      while (идент_ли(c) || !аски_ли(c) && юАльфа_ли())
      // Store identifier
      ткт_delim = начало[0..p-начало];
      // Scan нс
      if (сканируйНовСтр())
        --p; // Go back one because of "c = *++p;" in main loop.
      else
      {
        // TODO: ошибка(p, ИДС.ExpectedNewlineAfterIdentDelim);
      }
```

```
    }

    бул checkStringDelim(сим* р)
    {
      assert(ткт_delim.length != 0);
      if (буфер[$-1] == '\n' && // Last символ copied в буфер must be '\n'.
          конец-р >= ткт_delim.length && // Check remaining length.
          р[0..ткт_delim.length] == ткт_delim) // Compare.
        return да;
      return нет;
    }

    while (1)
    {
      с = *++р;
      switch (с)
      {
      case '\r':
        if (р[1] == '\n')
          ++р;
      case '\n':
        assert(конецНовСтроки_ли(р));
        с = '\n'; // Convert Новстр в '\n'.
        ++номСтр;
        установиНачалоСтроки(р+1);
        break;
      case 0, _Z_:
        // TODO: ошибка(tokenLineNum, tokenLineBegin, т.старт,
ИДС.UnterminatedDelimitedString);
        goto Lreturn3;
      default:
        if (!аски_ли(с))
        {
          auto начало = р;
          с = раскодируйЮ8();
          if (симНовСтрЮ_ли(с))
            goto case '\n';
          if (с == закрывающий_delim)
          {
            if (ткт_delim.length)
            {
              if (checkStringDelim(начало))
              {
                р = начало + ткт_delim.length;
                goto Lreturn2;
              }
            }
            else
            {
              assert(уровень == 1);
              --уровень;
              goto Lreturn;
            }
          }
          encodeUTF8(буфер, с);
          continue;
        }
        else
        {
          if (с == открывающий_delim)
            ++уровень;
          else if (с == закрывающий_delim)
          {
```

```
            if (ткт_delim.length)
            {
              if (checkStringDelim(p))
              {
                p += ткт_delim.length;
                goto Lreturn2;
              }
            }
            else if (--уровень == 0)
              goto Lreturn;
          }
        }
      }
      assert(аски_ли(c));
      буфер ~= c;
    }
  Lreturn: // Character delimiter.
    assert(c == закрывающий_delim);
    assert(уровень == 0);
    ++p; // Skip закрывающий delimiter.
  Lreturn2: // Ткст delimiter.
    if (*p == '"')
      ++p;
    else
    {
      // TODO: ошибка(p, ИДС.ExpectedDblQuoteAfterDelim, ткт_delim.length ?
ткт_delim : закрывающий_delim~"");
    }

    t.pf = scanPostfix();
  Lreturn3: // Ошибка.
    t.ткт = буфер ~ '\0';
    t.конец = p;
  }

  проц  scanTokenStringLiteral(ref Сема t)
  {
    assert(p[0] == 'q' && p[1] == '{');
    t.вид = ТОК.Ткст;

    auto tokenLineNum = номСтр;
    auto tokenLineBegin = началоСтроки;

    // A guard against changes в particular члены:
    // this.lineNum_hline and this.errorPath
    ++inTokenString;

    бцел номСтр = this.номСтр;
    бцел уровень = 1;

    ++p; ++p; // Skip q{

    auto предш_t = &t;
    Сема* сема;
    while (1)
    {
      сема = new Сема;
      сканируй(*сема);
      // Save the семы in a doubly linked список.
      // Could be useful for various tools.
      сема.предш = предш_t;
      предш_t.следщ = сема;
      предш_t = сема;
```

```
      switch (сема.вид)
      {
      case ТОК.ЛФСкобка:
        ++уровень;
        continue;
      case ТОК.ПФСкобка:
        if (--уровень == 0)
        {
          t.tok_ткт = t.следщ;
          t.следщ = null;
          break;
        }
        continue;
      case ТОК.КФ:
        // TODO: ошибка(tokenLineNum, tokenLineBegin, t.старт,
ИДС.UnterminatedTokenString);
        t.tok_ткт = t.следщ;
        t.следщ = сема;
        break;
      default:
        continue;
      }
      break; // Exit loop.
    }

    assert(сема.вид == ТОК.ПФСкобка || сема.вид == ТОК.КФ);
    assert(сема.вид == ТОК.ПФСкобка && t.следщ is null ||
           сема.вид == ТОК.КФ && t.следщ !is null);

    ткст буфер;
    // сема points в } or КФ
    if (сема.вид == ТОК.КФ)
    {
      t.конец = сема.старт;
      буфер = t.исхТекст[2..$].dup ~ '\0';
    }
    else
    {
      // Присвоить в буфер before scanPostfix().
      t.конец = p;
      буфер = t.исхТекст[2..$-1].dup ~ '\0';
      t.pf = scanPostfix();
      t.конец = p; // Присвоить again because of postfix.
    }
    // Convert newlines в '\n'.
    if (номСтр != this.номСтр)
    {
      assert(буфер[$-1] == '\0');
      бцел i, j;
      for (; i < буфер.length; ++i)
        switch (буфер[i])
        {
        case '\r':
          if (буфер[i+1] == '\n')
            ++i;
        case '\n':
          assert(конецНовСтроки_ли(буфер.ptr + i));
          буфер[j++] = '\n'; // Convert Новстр в '\n'.
          break;
        default:
          if (новСтрЮ_ли(буфер.ptr + i))
          {
            ++i; ++i;
```

```
            goto case '\n';
          }
          буфер[j++] = буфер[i]; // Copy.
        }
      буфер.length = j; // Adjust length.
    }
    assert(буфер[$-1] == '\0');
    t.ткт = буфер;

    --inTokenString;
  }
} // version(D2)

  /// Scans an escape sequence.
  ///
  /// EscapeSequence := "\" (Восмиричный{1,3} | ("x" Гекс{2}) |
  ///                        ("u" Гекс{4}) | ("U" Гекс{8}) |
  ///                        "'" | "\"" | "\\" | "?" | "a" |
  ///                        "b" | "f" | "n" | "r" | "t" | "v")
  /// Параметры:
  ///   isBinary = установи в да for octal and hexadecimal escapes.
  /// Возвращает: the escape значение.
  дим scanEscapeSequence(ref бул isBinary)
  out(результат)
  { assert(верноСимвол_ли(результат)); }
  body
  {
    assert(*p == '\\');

    auto sequenceStart = p; // Used for ошибка reporting.

    ++p;
    бцел c = сим8еск(*p);
    if (c)
    {
      ++p;
      return c;
    }

    бцел digits = 2;

    switch (*p)
    {
    case 'x':
      isBinary = да;
    case_Unicode:
      assert(c == 0);
      assert(digits == 2 || digits == 4 || digits == 8);
      while (1)
      {
        ++p;
        if (гекс_ли(*p))
        {
          c *= 16;
          if (*p <= '9')
            c += *p - '0';
          else if (*p <= 'F')
            c += *p - 'A' + 10;
          else
            c += *p - 'a' + 10;

          if (--digits == 0)
          {
```

```
            ++p;
            if (верноСимвол_ли(c))
              return c; // Итог valid escape значение.

            ошибка(sequenceStart, ИДС.InvalidUnicodeEscapeSequence,
                  sequenceStart[0..p-sequenceStart]);
            break;
          }
          continue;
        }

        ошибка(sequenceStart, ИДС.InsufficientHexDigits,
              sequenceStart[0..p-sequenceStart]);
        break;
      }
      break;
    case 'u':
      digits = 4;
      goto case_Unicode;
    case 'U':
      digits = 8;
      goto case_Unicode;
    default:
      if (восмир_ли(*p))
      {
        isBinary = да;
        assert(c == 0);
        c += *p - '0';
        ++p;
        if (!восмир_ли(*p))
          return c;
        c *= 8;
        c += *p - '0';
        ++p;
        if (!восмир_ли(*p))
          return c;
        c *= 8;
        c += *p - '0';
        ++p;
        if (c > 0xFF)
          ошибка(sequenceStart,
сооб.НевернаяВосмеричнаяПоследовательностьУклонения,
                sequenceStart[0..p-sequenceStart]);
        return c; // Итог valid escape значение.
      }
      else if(*p == '&')
      {
        if (буква_ли(*++p))
        {
          auto начало = p;
          while (цифробукв_ли(*++p))
          {}

          if (*p == ';')
          {
            // Pass сущность excluding '&' and ';'.
            c = сущностьВЮникод(начало[0..p - начало]);
            ++p; // Skip ;
            if (c != 0xFFFF)
              return c; // Итог valid escape значение.
            else
              ошибка(sequenceStart, ИДС.UndefinedHTMLEntity, sequenceStart[0
.. p - sequenceStart]);
```

```
          }
          else
            ошибка(sequenceStart, ИДС.UnterminatedHTMLEntity, sequenceStart[0
.. p - sequenceStart]);
        }
        else
          ошибка(sequenceStart, ИДС.InvalidBeginHTMLEntity);
      }
      else if (конецСтроки_ли(p))
        ошибка(sequenceStart, ИДС.UndefinedEscapeSequence,
          кф_ли(*p) ? `\КФ` : `\NewLine`);
      else
      {
        ткст ткт = `\`;
        if (аски_ли(c))
          ткт ~= *p;
        else
          encodeUTF8(ткт, раскодируйЮ8());
        ++p;
        // TODO: check for unprintable символ?
        ошибка(sequenceStart, ИДС.UndefinedEscapeSequence, ткт);
      }
    }
    return СИМ_ЗАМЕНЫ; // Ошибка: return replacement символ.
  }

  /// Scans a число literal.
  ///
  /// $(PRE
  /// IntegerLiteral := (Dec|Гекс|Bin|Oct)Suffix?
  /// Dec := (0|[1-9][0-9_]*)
  /// Гекс := 0[xX][_]*[0-9a-zA-Z][0-9a-zA-Z_]*
  /// Bin := 0[bB][_]*[01][01_]*
  /// Oct := 0[0-7_]*
  /// Suffix := (L[uU]?|[uU]L?)
  /// )
  /// Неверно: "0b_", "0x_", "._" etc.
  проц  scanNumber(ref Сема t)
  {
    бдол бдол_;
    бул overflow;
    бул isDecimal;
    т_мера digits;

    if (*p != '0')
      goto LscanInteger;
    ++p; // пропусти zero
    // check for xX bB ...
    switch (*p)
    {
    case 'x','X':
      goto LscanHex;
    case 'b','B':
      goto LscanBinary;
    case 'L':
      if (p[1] == 'i')
        goto LscanReal; // 0Li
      break; // 0L
    case '.':
      if (p[1] == '.')
        break; // 0..
      // 0.
    case 'i','f','F', // Мнимое and плав literal suffixes.
```

```
        'e', 'E':    // Плав exponent.
      goto LscanReal;
    default:
      if (*p == '_')
        goto LscanOctal; // 0_
      else if (цифра_ли(*p))
      {
        if (*p == '8' || *p == '9')
          goto Loctal_hasDecimalDigits; // 08 or 09
        else
          goto Loctal_enter_loop; // 0[0-7]
      }
    }

    // Число 0
    assert(p[-1] == '0');
    assert(*p != '_' && !цифра_ли(*p));
    assert(бдол_ == 0);
    isDecimal = да;
    goto Lfinalize;

  LscanInteger:
    assert(*p != 0 && цифра_ли(*p));
    isDecimal = да;
    goto Lenter_loop_int;
    while (1)
    {
      if (*++p == '_')
        continue;
      if (!цифра_ли(*p))
        break;
    Lenter_loop_int:
      if (бдол_ < бдол.max/10 || (бдол_ == бдол.max/10 && *p <= '5'))
      {
        бдол_ *= 10;
        бдол_ += *p - '0';
        continue;
      }
      // Overflow: пропусти following digits.
      overflow = да;
      while (цифра_ли(*++p)) {}
      break;
    }

    // The число could be a плав, so check overflow below.
    switch (*p)
    {
    case '.':
      if (p[1] != '.')
        goto LscanReal;
      break;
    case 'L':
      if (p[1] != 'i')
        break;
    case 'i', 'f', 'F', 'e', 'E':
      goto LscanReal;
    default:
    }

    if (overflow)
      ошибка(t.старт, ИДС.OverflowDecimalNumber);

    assert((цифра_ли(p[-1]) || p[-1] == '_') && !цифра_ли(*p) && *p != '_');
```

```
    goto Lfinalize;

  LscanHex:
    assert(digits == 0);
    assert(*p == 'x' || *p == 'X');
    while (1)
    {
      if (*++p == '_')
        continue;
      if (!гекс_ли(*p))
        break;
      ++digits;
      бдол_ *= 16;
      if (*p <= '9')
        бдол_ += *p - '0';
      else if (*p <= 'F')
        бдол_ += *p - 'A' + 10;
      else
        бдол_ += *p - 'a' + 10;
    }

    assert(гекс_ли(p[-1]) || p[-1] == '_' || p[-1] == 'x' || p[-1] == 'X');
    assert(!гекс_ли(*p) && *p != '_');

    switch (*p)
    {
    case '.':
      if (p[1] == '.')
        break;
    case 'p', 'P':
      return scanHexReal(t);
    default:
    }

    if (digits == 0 || digits > 16)
      ошибка(t.старт, digits == 0 ? ИДС.NoDigitsInHexNumber :
ИДС.OverflowHexNumber);

    goto Lfinalize;

  LscanBinary:
    assert(digits == 0);
    assert(*p == 'b' || *p == 'B');
    while (1)
    {
      if (*++p == '0')
      {
        ++digits;
        бдол_ *= 2;
      }
      else if (*p == '1')
      {
        ++digits;
        бдол_ *= 2;
        бдол_ += *p - '0';
      }
      else if (*p == '_')
        continue;
      else
        break;
    }

    if (digits == 0 || digits > 64)
```

```
      ошибка(t.старт, digits == 0 ? ИДС.NoDigitsInBinNumber :
ИДС.OverflowBinaryNumber);

    assert(p[-1] == '0' || p[-1] == '1' || p[-1] == '_' || p[-1] == 'b' ||
p[-1] == 'B', p[-1] ~ "");
    assert( !(*p == '0' || *p == '1' || *p == '_') );
    goto Lfinalize;

  LscanOctal:
    assert(*p == '_');
    while (1)
    {
      if (*++p == '_')
        continue;
      if (!восмир_ли(*p))
        break;
    Loctal_enter_loop:
      if (бдол_ < бдол.max/2 || (бдол_ == бдол.max/2 && *p <= '1'))
      {
        бдол_ *= 8;
        бдол_ += *p - '0';
        continue;
      }
      // Overflow: пропусти following digits.
      overflow = да;
      while (восмир_ли(*++p)) {}
      break;
    }

    бул hasDecimalDigits;
    if (цифра_ли(*p))
    {
    Loctal_hasDecimalDigits:
      hasDecimalDigits = да;
      while (цифра_ли(*++p)) {}
    }

    // The число could be a плав, so check ошибки below.
    switch (*p)
    {
    case '.':
      if (p[1] != '.')
        goto LscanReal;
      break;
    case 'L':
      if (p[1] != 'i')
        break;
    case 'i', 'f', 'F', 'e', 'E':
      goto LscanReal;
    default:
    }

    if (hasDecimalDigits)
      ошибка(t.старт, ИДС.OctalNumberHasDecimals);

    if (overflow)
      ошибка(t.старт, ИДС.OverflowOctalNumber);
//     goto Lfinalize;

  Lfinalize:
    enum Suffix
    {
      Нет     = 0,
```

```
  Unsigned = 1,
  Дол      = 2
}

// Scan optional суффикс: L, Lu, LU, u, uL, U or UL.
Suffix суффикс;
while (1)
{
  switch (*p)
  {
  case 'L':
    if (суффикс & Suffix.Дол)
      break;
    суффикс |= Suffix.Дол;
    ++p;
    continue;
  case 'u', 'U':
    if (суффикс & Suffix.Unsigned)
      break;
    суффикс |= Suffix.Unsigned;
    ++p;
    continue;
  default:
    break;
  }
  break;
}

// Determine тип of Integer.
switch (суффикс)
{
case Suffix.Нет:
  if (бдол_ & 0x8000_0000_0000_0000)
  {
    if (isDecimal)
      ошибка(t.старт, ИДС.OverflowDecimalSign);
    t.вид = ТОК.Бцел64;
  }
  else if (бдол_ & 0xFFFF_FFFF_0000_0000)
    t.вид = ТОК.Цел64;
  else if (бдол_ & 0x8000_0000)
    t.вид = isDecimal ? ТОК.Цел64 : ТОК.Бцел32;
  else
    t.вид = ТОК.Цел32;
  break;
case Suffix.Unsigned:
  if (бдол_ & 0xFFFF_FFFF_0000_0000)
    t.вид = ТОК.Бцел64;
  else
    t.вид = ТОК.Бцел32;
  break;
case Suffix.Дол:
  if (бдол_ & 0x8000_0000_0000_0000)
  {
    if (isDecimal)
      ошибка(t.старт, ИДС.OverflowDecimalSign);
    t.вид = ТОК.Бцел64;
  }
  else
    t.вид = ТОК.Цел64;
  break;
case Suffix.Unsigned | Suffix.Дол:
  t.вид = ТОК.Бцел64;
```

```
    break;
  default:
    assert(0);
  }
  t.бдол_ = бдол_;
  t.конец = p;
  return;
LscanReal:
  scanReal(t);
  return;
}

/// Scans a floating point число literal.
///
/// $(PRE
/// ПлавLiteral := Плав[fFL]?i?
/// Плав := DecПлав | HexПлав
/// DecПлав := ([0-9][0-9_]*[.][0-9_]*DecExponent?) |
///             [.][0-9][0-9_]*DecExponent? | [0-9][0-9_]*DecExponent
/// DecExponent := [eE][+-]?[0-9][0-9_]*
/// HexПлав := 0[xX](HexDigits[.]HexDigits |
///                  [.][0-9a-zA-Z]HexDigits? |
///                  HexDigits)HexExponent
/// HexExponent := [pP][+-]?[0-9][0-9_]*
/// )
проц  scanReal(ref Сема t)
{
  if (*p == '.')
  {
    assert(p[1] != '.');
    // Этот function was called by сканируй() or scanNumber().
    while (цифра_ли(*++p) || *p == '_') {}
  }
  else
    // Этот function was called by scanNumber().
    assert(delegate ()
      {
        switch (*p)
        {
        case 'L':
          if (p[1] != 'i')
            return нет;
        case 'i', 'f', 'F', 'e', 'E':
          return да;
        default:
        }
        return нет;
      }()
    );

  // Scan exponent.
  if (*p == 'e' || *p == 'E')
  {
    ++p;
    if (*p == '-' || *p == '+')
      ++p;
    if (цифра_ли(*p))
      while (цифра_ли(*++p) || *p == '_') {}
    else
      ошибка(t.старт, ИДС.ПлавExpMustStartWithDigit);
  }

  // Copy whole число and remove underscores из буфер.
```

```
  ткст буфер = t.старт[0..p-t.старт].dup;
  бцел j;
  foreach (c; буфер)
    if (c != '_')
      буфер[j++] = c;
  буфер.length = j; // Adjust length.
  буфер ~= 0; // Terminate for C functions.

  finalizeПлав(t, буфер);
}

/// Scans a hexadecimal floating point число literal.
проц  scanHexReal(ref Сема t)
{
  assert(*p == '.' || *p == 'p' || *p == 'P');
  ИДС идс;
  if (*p == '.')
    while (гекс_ли(*++p) || *p == '_')
    {}
  // Decimal exponent is required.
  if (*p != 'p' && *p != 'P')
  {
    идс = ИДС.НехПлавExponentRequired;
    goto Lerr;
  }
  // Scan exponent
  assert(*p == 'p' || *p == 'P');
  ++p;
  if (*p == '+' || *p == '-')
    ++p;
  if (!цифра_ли(*p))
  {
    идс = ИДС.НехПлавExpMustStartWithDigit;
    goto Lerr;
  }
  while (цифра_ли(*++p) || *p == '_')
  {}
  // Copy whole число and remove underscores из буфер.
  ткст буфер = t.старт[0..p-t.старт].dup;
  бцел j;
  foreach (c; буфер)
    if (c != '_')
      буфер[j++] = c;
  буфер.length = j; // Adjust length.
  буфер ~= 0; // Terminate for C functions.
  finalizeПлав(t, буфер);
  return;
Lerr:
  t.вид = ТОК.Плав32;
  t.конец = p;
  ошибка(t.старт, идс);
}

/// Sets the значение of the сема.
/// Параметры:
///   t = receives the значение.
///   буфер = the well-formed плав число.
проц  finalizeПлав(ref Сема t, ткст буфер)
{
  assert(буфер[$-1] == 0);
  // Плав число is well-formed. Check suffixes and do conversion.
  switch (*p)
  {
```

```
  case 'f', 'F':
    t.вид = ТОК.Плав32;
    t.плав_ = strtof(буфер.ptr, null);
    ++p;
    break;
  case 'L':
    t.вид = ТОК.Плав80;
    t.реал_ = strtold(буфер.ptr, null);
    ++p;
    break;
  default:
    t.вид = ТОК.Плав64;
    t.дво_ = strtod(буфер.ptr, null);
  }
  if (*p == 'i')
  {
    ++p;
    t.вид += 3; // Щит в imaginary counterpart.
    assert(t.вид == ТОК.Мнимое32 ||
           t.вид == ТОК.Мнимое64 ||
           t.вид == ТОК.Мнимое80);
  }
  if (errno() == ERANGE)
    ошибка(t.старт, ИДС.OverflowПлавNumber);
  t.конец = p;
}

/// Scans a special сема sequence.
///
/// SpecialTokenSequence := "#line" Integer Filespec? EndOfLine
проц  scanSpecialTokenSequence(ref Сема t)
{
  assert(*p == '#');
  t.вид = ТОК.HashLine;
  t.установиФлагПробельные();

  ИДС идс;
  сим* errorAtColumn = p;
  сим* tokenEnd = ++p;

  if (!(p[0] == 'l' && p[1] == 'i' && p[2] == 'n' && p[3] == 'e'))
  {
    идс = ИДС.ExpectedIdentifierSTLine;
    goto Lerr;
  }
  p += 3;
  tokenEnd = p + 1;

  // TODO: #line58"путь/file" is legal. Require spaces?
  //       State.Space could be used for that purpose.
  enum State
  { /+Space,+/ Integer, Filespec, End }

  State state = State.Integer;

  while (!конецСтроки_ли(++p))
  {
    if (пбел_ли(*p))
      continue;
    if (state == State.Integer)
    {
      if (!цифра_ли(*p))
      {
```

```
        errorAtColumn = p;
        идс = ИДС.ExpectedIntegerAfterSTLine;
        goto Lerr;
      }
      t.tokLineNum = new Сема;
      сканируй(*t.tokLineNum);
      tokenEnd = p;
      if (t.tokLineNum.вид != ТОК.Цел32 && t.tokLineNum.вид != ТОК.Бцел32)
      {
        errorAtColumn = t.tokLineNum.старт;
        идс = ИДС.ExpectedIntegerAfterSTLine;
        goto Lerr;
      }
      --p; // Go one back because сканируй() advanced p past the integer.
      state = State.Filespec;
    }
    else if (state == State.Filespec && *p == '"')
    { // ИДС.ExpectedFilespec is deprecated.
      // if (*p != '"')
      // {
      //   errorAtColumn = p;
      //   идс = ИДС.ExpectedFilespec;
      //   goto Lerr;
      // }
      t.tokLineFilespec = new Сема;
      t.tokLineFilespec.старт = p;
      t.tokLineFilespec.вид = ТОК.Filespec;
      t.tokLineFilespec.установиФлагПробельные();
      while (*++p != '"')
      {
        if (конецСтроки_ли(p))
        {
          errorAtColumn = t.tokLineFilespec.старт;
          идс = ИДС.НеоконченоеУказаниеФайла;
          t.tokLineFilespec.конец = p;
          tokenEnd = p;
          goto Lerr;
        }
        аски_ли(*p) || раскодируйЮ8();
      }
      auto старт = t.tokLineFilespec.старт +1; // +1 пропустив '"'
      t.tokLineFilespec.ткт = старт[0 .. p - старт];
      t.tokLineFilespec.конец = p + 1;
      tokenEnd = p + 1;
      state = State.End;
    }
    else/+ if (state == State.End)+/
    {
      идс = ИДС.НеоконченыйОсобыйТокен;
      goto Lerr;
    }
  }
  assert(конецСтроки_ли(p));

  if (state == State.Integer)
  {
    errorAtColumn = p;
    идс = ИДС.ExpectedIntegerAfterSTLine;
    goto Lerr;
  }

  // Evaluate #line only when not in сема ткст.
  if (!inTokenString && t.tokLineNum)
```

```
    {
      this.lineNum_hline = this.номСтр - t.tokLineNum.бцел_ + 1;
      if (t.tokLineFilespec)
        новыйПутьФ(t.tokLineFilespec.ткт);
    }
    p = tokenEnd;
    t.конец = tokenEnd;

    return;
  Lerr:
    p = tokenEnd;
    t.конец = tokenEnd;
    ошибка(errorAtColumn, идс);
  }

  /// Inserts an empty dummy сема (ТОК.Пусто) before t.
  ///
  /// Useful in the parsing phase for representing a узел in the AST
  /// that doesn't consume an actual сема из the source текст.
  Сема* вставьПустуюСемуПеред(Сема* t)
  {
    assert(t !is null && t.предш !is null);
    assert(текст.ptr <= t.старт && t.старт < конец, Сема.вТкст(t.вид));
    assert(текст.ptr <= t.конец && t.конец <= конец, Сема.вТкст(t.вид));

    auto предш_t = t.предш;
    auto т_нов = new Сема;
    т_нов.вид = ТОК.Пусто;
    т_нов.старт = т_нов.конец = предш_t.конец;
    // Link in new сема.
    предш_t.следщ = т_нов;
    т_нов.предш = предш_t;
    т_нов.следщ = t;
    t.предш = т_нов;
    return т_нов;
  }

  /// Возвращает ошибка line число.
  бцел номерСтрокиОшиб(бцел номСтр)
  {
    return номСтр - this.lineNum_hline;
  }

  /// Forwards ошибка параметры.
  проц  ошибка(сим* columnPos, ткст сооб, ...)
  {
    error_(this.номСтр, this.началоСтроки, columnPos, сооб, _arguments,
_argptr);
  }

  /// определено
  проц  ошибка(сим* columnPos, ИДС идс, ...)
  {
    error_(this.номСтр, this.началоСтроки, columnPos, ДайСооб(идс),
_arguments, _argptr);
  }

  /// определено
  проц  ошибка(бцел номСтр, сим* началоСтроки, сим* columnPos, ИДС идс, ...)
  {
    error_(номСтр, началоСтроки, columnPos, ДайСооб(идс), _arguments,
_argptr);
  }
```

```
  /// Creates an ошибка report and appends it в a список.
  /// Параметры:
  ///   номСтр = the line число.
  ///   началоСтроки = points в the first символ of the current line.
  ///   columnPos = points в the символ where the ошибка is located.
  ///   сооб = the сообщение.
  проц  error_(бцел номСтр, сим* началоСтроки, сим* columnPos, ткст сооб,
               TypeInfo[] _arguments, base.спис_ва _argptr)
  {
    номСтр = this.номерСтрокиОшиб(номСтр);
    auto errorPath = this.путиКФайлам.устПуть;
    auto положение = new Положение(errorPath, номСтр, началоСтроки,
columnPos);
    сооб = Формат(_arguments, _argptr, сооб);
    auto ошибка = new ОшибкаЛексера(положение, сооб);
    ошибки ~= ошибка;
    if (диаг !is null)
      диаг ~= ошибка;
  }

  /// Scans the whole source текст until КФ is encountered.
  проц  сканируйВсе()
  {
    while (следщСема() != ТОК.КФ)
    {}
  }

  /// Возвращает first сема of the source текст.
  /// Этот can be the КФ сема.
  /// Structure: ГОЛОВА -> Новстр -> First Сема
  Сема* перваяСема()
  {
    return this.глава.следщ.следщ;
  }

  /// Returns да if ткт is a valid D identifier.
  static бул строкаИдентификатора_ли(ткст ткт)
  {
    if (ткт.length == 0 || цифра_ли(ткт[0]))
      return нет;
    т_мера idx;
    do
    {
      auto c = drc.Unicode.раскодируй(ткт, idx);
      if (c == СИМ_ОШИБКИ || !(идент_ли(c) || !аски_ли(c) && униАльфа_ли(c)))
        return нет;
    } while (idx < ткт.length)
    return да;
  }

  /// Returns да if ткт is a keyword or
  /// a special сема (__FILE__, __LINE__ etc.)
  static бул резервныйИдентификатор_ли(ткст ткт)
  {
    if (ткт.length == 0)
      return нет;
    auto ид = ТаблицаИд.вСтатической(ткт);
    if (ид is null || ид.вид == ТОК.Идентификатор)
      return нет; // ткт is not in the таблица or a normal identifier.
    return да;
  }
```

```
  /// Возвращает да, если из_ a valid identifier and if it's not reserved.
  static бул действитНерезИдентификатор_ли(ткст ткт)
  {
    return строкаИдентификатора_ли(ткт) && !резервныйИдентификатор_ли(ткт);
  }

  /// Returns да if the current символ в be decoded is
  /// a Unicode alpha символ.
  ///
  /// The current pointer 'p' is установи в the last trailbyte if да is
returned.
  бул юАльфа_ли()
  {
    assert(!аски_ли(*p), "check for ASCII сим before calling
раскодируйЮ8().");
    сим* p = this.p;
    дим d = *p;
    ++p; // Move в second байт.
    // Ошибка if second байт is not a trail байт.
    if (!ведомыйБайт_ли(*p))
      return нет;
    // Check for overlong sequences.
    switch (d)
    {
    case 0xE0, 0xF0, 0xF8, 0xFC:
      if ((*p & d) == 0x80)
        return нет;
    default:
      if ((d & 0xFE) == 0xC0) // 1100000x
        return нет;
    }
    const ткст проверьСледующийБайт = "if (!ведомыйБайт_ли(*++p))"
                                       "  return нет;";
    const ткст добавьШестьБит = "d = (d << 6) | *p & 0b0011_1111;";
    // Decode
    if ((d & 0b1110_0000) == 0b1100_0000)
    {
      d &= 0b0001_1111;
      mixin(добавьШестьБит);
    }
    else if ((d & 0b1111_0000) == 0b1110_0000)
    {
      d &= 0b0000_1111;
      mixin(добавьШестьБит ~
            проверьСледующийБайт ~ добавьШестьБит);
    }
    else if ((d & 0b1111_1000) == 0b1111_0000)
    {
      d &= 0b0000_0111;
      mixin(добавьШестьБит ~
            проверьСледующийБайт ~ добавьШестьБит ~
            проверьСледующийБайт ~ добавьШестьБит);
    }
    else
      return нет;

    assert(ведомыйБайт_ли(*p));
    if (!верноСимвол_ли(d) || !униАльфа_ли(d))
      return нет;
    // Only advance pointer if this is a Unicode alpha символ.
    this.p = p;
    return да;
  }
```

```
  /// Decodes the следщ UTF-8 sequence.
  дим раскодируйЮ8()
  {
    assert(!аски_ли(*p), "check for ASCII char before calling
раскодируйЮ8().");
    сим* p = this.p;
    дим d = *p;

    ++p; // Move в second байт.
    // Ошибка if second байт is not a trail байт.
    if (!ведомыйБайт_ли(*p))
      goto Lerr2;

    // Check for overlong sequences.
    switch (d)
    {
    case 0xE0, // 11100000 100xxxxx
         0xF0, // 11110000 1000xxxx
         0xF8, // 11111000 10000xxx
         0xFC: // 11111100 100000xx
      if ((*p & d) == 0x80)
        goto Lerr;
    default:
      if ((d & 0xFE) == 0xC0) // 1100000x
        goto Lerr;
    }

    const ткст проверьСледующийБайт = "if (!ведомыйБайт_ли(*++p))"
                                       "  goto Lerr2;";
    const ткст добавьШестьБит = "d = (d << 6) | *p & 0b0011_1111;";

    // Decode
    if ((d & 0b1110_0000) == 0b1100_0000)
    { // 110xxxxx 10xxxxxx
      d &= 0b0001_1111;
      mixin(добавьШестьБит);
    }
    else if ((d & 0b1111_0000) == 0b1110_0000)
    { // 1110xxxx 10xxxxxx 10xxxxxx
      d &= 0b0000_1111;
      mixin(добавьШестьБит ~
            проверьСледующийБайт ~ добавьШестьБит);
    }
    else if ((d & 0b1111_1000) == 0b1111_0000)
    { // 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
      d &= 0b0000_0111;
      mixin(добавьШестьБит ~
            проверьСледующийБайт ~ добавьШестьБит ~
            проверьСледующийБайт ~ добавьШестьБит);
    }
    else
      // 5 and 6 байт UTF-8 sequences are not allowed yet.
      // 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
      // 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
      goto Lerr;

    assert(ведомыйБайт_ли(*p));

    if (!верноСимвол_ли(d))
    {
    Lerr:
      // Three cases:
```

```
      // *) the UTF-8 sequence was successfully decoded but the resulting
      //    символ is invalid.
      //    p points в last trail байт in the sequence.
      // *) the UTF-8 sequence is overlong.
      //    p points в second байт in the sequence.
      // *) the UTF-8 sequence has more than 4 bytes or starts with
      //    a trail байт.
      //    p points в second байт in the sequence.
      assert(ведомыйБайт_ли(*p));
      // Move в следщ ASCII символ or lead байт of a UTF-8 sequence.
      while (p < (конец-1) && ведомыйБайт_ли(*p))
        ++p;
      --p;
      assert(!ведомыйБайт_ли(p[1]));
    Lerr2:
      d = СИМ_ЗАМЕНЫ;
      ошибка(this.p, ИДС.НедействительнаяПоследовательностьУТФ8,
formatBytes(this.p, p));
    }

    this.p = p;
    return d;
  }

  /// Encodes the символ d and appends it в ткт.
  static проц  encodeUTF8(ref ткст ткт, дим d)
  {
    assert(!аски_ли(d), "check for ASCII сим before calling encodeUTF8().");
    assert(верноСимвол_ли(d), "check if символ is valid before calling
encodeUTF8().");

    сим[6] b = void;
    if (d < 0x800)
    {
      b[0] = 0xC0 | (d >> 6);
      b[1] = 0x80 | (d & 0x3F);
      ткт ~= b[0..2];
    }
    else if (d < 0x10000)
    {
      b[0] = 0xE0 | (d >> 12);
      b[1] = 0x80 | ((d >> 6) & 0x3F);
      b[2] = 0x80 | (d & 0x3F);
      ткт ~= b[0..3];
    }
    else if (d < 0x200000)
    {
      b[0] = 0xF0 | (d >> 18);
      b[1] = 0x80 | ((d >> 12) & 0x3F);
      b[2] = 0x80 | ((d >> 6) & 0x3F);
      b[3] = 0x80 | (d & 0x3F);
      ткт ~= b[0..4];
    }
    /+ // There are no 5 and 6 байт UTF-8 sequences yet.
    else if (d < 0x4000000)
    {
      b[0] = 0xF8 | (d >> 24);
      b[1] = 0x80 | ((d >> 18) & 0x3F);
      b[2] = 0x80 | ((d >> 12) & 0x3F);
      b[3] = 0x80 | ((d >> 6) & 0x3F);
      b[4] = 0x80 | (d & 0x3F);
      ткт ~= b[0..5];
    }
```

```
    else if (d < 0x80000000)
    {
      b[0] = 0xFC | (d >> 30);
      b[1] = 0x80 | ((d >> 24) & 0x3F);
      b[2] = 0x80 | ((d >> 18) & 0x3F);
      b[3] = 0x80 | ((d >> 12) & 0x3F);
      b[4] = 0x80 | ((d >> 6) & 0x3F);
      b[5] = 0x80 | (d & 0x3F);
      ткт ~= b[0..6];
    }
    +/
    else
     assert(0);
  }

  /// Formats the bytes between старт and конец.
  /// Возвращает: в.g.: abc -> \x61\x62\x63
  static ткст formatBytes(сим* старт, сим* конец)
  {
    auto ткстLen = конец-старт;
    const formatLen = `\xXX`.length;
    ткст результат = new сим[ткстLen*formatLen]; // Reserve space.
    результат.length = 0;
    foreach (с; cast(ббайт[])старт[0..ткстLen])
      результат ~= Формат("\\x{:X}", с);
    return результат;
  }

  /// Searches for an invalid UTF-8 sequence in ткт.
  /// Возвращает: a formatted ткст of the invalid sequence (в.g. \xC0\x80).
  static ткст найдиНедействительнуюПоследовательностьУТФ8(ткст ткт)
  {
    сим* р = ткт.ptr, конец = р + ткт.length;
    while (р < конец)
    {
      if (раскодируй(р, конец) == СИМ_ОШИБКИ)
      {
        auto начало = р;
        // Skip trail-bytes.
        while (++р < конец && ведомыйБайт_ли(*р))
        {}
        return Лексер.formatBytes(начало, р);
      }
    }
    assert(р == конец);
    return "";
  }
}

/// Tests the лексер with a список of семы.
unittest
{
  выдай("Тестируем Лексер.\n");
  struct Пара
  {
    ткст текстТокена;
    ТОК вид;
  }
  static Пара[] пары = [
    {"#!äöüß",  ТОК.Шебанг},       {"\n",      ТОК.Новстр},
    {"//çay",   ТОК.Комментарий},      {"\n",      ТОК.Новстр},
                                    {"&",       ТОК.ИБинарное},
    {"/*çağ*/", ТОК.Комментарий},      {"&&",      ТОК.ИЛогическое},
```

```
    {"/+çak+/", ТОК.Комментарий},       {"&=",      ТОК.ИПрисвой},
    {">",       ТОК.Больше},        {"+",       ТОК.Плюс},
    {">=",      ТОК.БольшеРавно},   {"++",      ТОК.ПлюсПлюс},
    {">>",      ТОК.ПСдвиг},         {"+=",      ТОК.ПлюсПрисвой},
    {">>=",     ТОК.ПСдвигПрисвой},   {"-",       ТОК.Минус},
    {">>>",     ТОК.URShift},        {"--",      ТОК.МинусМинус},
    {">>>=",    ТОК.URShiftAssign}, {"-=",      ТОК.МинусПрисвой},
    {"<",       ТОК.Меньше},          {"=",       ТОК.Присвоить},
    {"<=",      ТОК.МеньшеРавно},     {"==",      ТОК.Равно},
    {"<>",      ТОК.LorG},          {"~",       ТОК.Тильда},
    {"<>=",     ТОК.LorEorG},       {"~=",      ТОК.CatAssign},
    {"<<",      ТОК.ЛСдвиг},         {"*",       ТОК.Умножь},
    {"<<=",     ТОК.ЛСдвигПрисвой},   {"*=",      ТОК.УмножьПрисвой},
    {"!",       ТОК.Не},            {"/",       ТОК.Деление},
    {"!=",      ТОК.НеРавно},       {"/=",      ТОК.ДелениеПрисвой},
    {"!<",      ТОК.UorGorE},        {"^",       ТОК.ИИли},
    {"!>",      ТОК.UorLorE},        {"^=",      ТОК.ИИлиПрисвой},
    {"!<=",     ТОК.UorG},           {"%",       ТОК.Модуль},
    {"!>=",     ТОК.UorL},           {"%=",      ТОК.МодульПрисвой},
    {"!<>",     ТОК.UorE},           {"(",       ТОК.ЛСкобка},
    {"!<>=",    ТОК.Unordered},      {")",       ТОК.ПСкобка},
    {".",       ТОК.Точка},           {"[",       ТОК.ЛКвСкобка},
    {"..",      ТОК.Срез},          {"]",       ТОК.ПКвСкобка},
    {"...",     ТОК.Эллипсис},       {"{",       ТОК.ЛФСкобка},
    {"|",       ТОК.ИлиБинарное},      {"}",       ТОК.ПФСкобка},
    {"||",      ТОК.ИлиЛогическое},     {":",       ТОК.Двоеточие},
    {"|=",      ТОК.ИлиПрисвой},      {";",       ТОК.ТочкаЗапятая},
    {"?",       ТОК.Вопрос},      {",",       ТОК.Запятая},
    {"$",       ТОК.Доллар},         {"cam",     ТОК.Идентификатор},
    {"çay",     ТОК.Идентификатор},    {".0",      ТОК.Плав64},
    {"0",       ТОК.Цел32},          {"\n",      ТОК.Новстр},
    {"\r",      ТОК.Новстр},        {"\r\n",    ТОК.Новстр},
    {"\u2028",  ТОК.Новстр},        {"\u2029",  ТОК.Новстр}
  ];

  ткст ист;

  // Join all сема тексты into a single ткст.
  foreach (i, пара; пары)
    if (пара.вид == ТОК.Комментарий && пара.текстТокена[1] == '/' || // Line
comment.
        пара.вид == ТОК.Шебанг)
    {
      assert(пары[i+1].вид == ТОК.Новстр); // Must be followed by a нс.
      ист ~= пара.текстТокена;
    }
    else
      ист ~= пара.текстТокена ~ " ";

  // Lex the constructed source текст.
  auto lx = new Лексер(new ИсходныйТекст("", ист));
  lx.сканируйВсе();

  auto сема = lx.перваяСема();

  for (бцел i; i < пары.length && сема.вид != ТОК.КФ;
       ++i, (сема = сема.следщ))
    if (сема.исхТекст != пары[i].текстТокена)
      assert(0, Формат("Найдено '{0}' , но ожидалось '{1}'",
                       сема.исхТекст, пары[i].текстТокена));
}

/// Tests the Лексер's возьми() method.
```

```
unittest
{
  выдай("Тестируем метод Лексер.возьми()\n");
  auto исходныйТекст = new ИсходныйТекст("", "unittest { }");
  auto lx = new Лексер(исходныйТекст, null);

  auto следщ = lx.глава;
  lx.возьми(следщ);
  assert(следщ.вид == ТОК.Новстр);
  lx.возьми(следщ);
  assert(следщ.вид == ТОК.Юниттест);
  lx.возьми(следщ);
  assert(следщ.вид == ТОК.ЛФСкобка);
  lx.возьми(следщ);
  assert(следщ.вид == ТОК.ПФСкобка);
  lx.возьми(следщ);
  assert(следщ.вид == ТОК.КФ);

  lx = new Лексер(new ИсходныйТекст("", ""));
  следщ = lx.глава;
  lx.возьми(следщ);
  assert(следщ.вид == ТОК.Новстр);
  lx.возьми(следщ);
  assert(следщ.вид == ТОК.КФ);
}

unittest
{
  // Numbers unittest
  // 0L 0ULi 0_L 0_UL 0x0U 0x0p2 0_Fi 0_e2 0_F 0_i
  // 0u 0U 0uL 0UL 0L 0LU 0Lu
  // 0Li 0f 0F 0fi 0Fi 0i
  // 0b_1_LU 0b1000u
  // 0x232Lu
}
```

---

module drc.lexer.Token;

```
import drc.lexer.Identifier,
       drc.lexer.Funcs;
import drc.Location;
import cidrus : malloc, free;
import exception;
import common;

public import drc.lexer.TokensEnum;

/// А Сема - из_ цепочка символов, формируемая лексическим анализатором.
struct Сема
{ /// Флаги, устанавливаемые Лексером.
  enum Флаги : бкрат
  {
    Нет,
    Пробельный = 1, /// Знаки с этим флагом игнорируются Парсером.
  }

  ТОК вид; /// Вид семы.
  Флаги флаги; /// Флаги семы.
  /// Указатели на следующую и предыдущую семы (дважды линкованный список.)
  Сема* следщ, предш;

  /// Начало пробельных символов перед семой. Нуль, если их нет.
```

```
  /// TODO: remove в save space; can be replaced by 'предш.конец'.
  сим* пп;
  сим* старт; /// Указывает на первый символ семы.
  сим* конец;   /// Points one символ past the конец of the сема.

  /// Данные, ассоциированные с данной семой.
  /// TODO: move данные structures out; use only pointers here в keep
Сема.sizeof small.
  union
  {
    /// При нс семы.
    ДанныеНовСтр нс;
    /// При #line семы.
    struct
    {
      Сема* tokLineNum; /// #line число
      Сема* tokLineFilespec; /// #line число filespec
    }
    /// The значение of a ткст сема.
    struct
    {
      ткст ткт; /// Zero-terminated ткст. (The zero is included in the
length.)
      сим pf;    /// Postfix 'c', 'w', 'd' or 0 for none.
    version(D2)
      Сема* tok_ткт; /++ Points в the contents of a сема ткст stored as a
                        doubly linked список. The last сема is always '}'
or
                        КФ in case конец of source текст is "q{" КФ.
                     +/
    }
    Идентификатор* идент; /// При keywords and identifiers.
    дим  дим_;   /// A символ значение.
    дол   дол_;    /// A дол integer значение.
    бдол  бдол_;   /// An unsigned дол integer значение.
    цел    цел_;     /// An integer значение.
    бцел   бцел_;    /// An unsigned integer значение.
    плав  плав_;   /// A плав значение.
    дво дво_;  /// A дво значение.
    реал   реал_;    /// A реал значение.
  }

  /// Возвращает текст of the сема.
  ткст исхТекст()
  {
    assert(старт && конец);
    return старт[0 .. конец - старт];
  }

  /// Возвращает preceding whitespace of the сема.
  ткст пробСимволы()
  {
    assert(пп && старт);
    return пп[0 .. старт - пп];
  }

  /// Finds the следщ non-whitespace сема.
  /// Возвращает: 'эту' сему, если предыдущая является ТОК.ГОЛОВА или null.
  Сема* следщНепроб()
  out(сема)
  {
    assert(сема !is null);
  }
```

```
  body
  {
    auto сема = следщ;
    while (сема !is null && сема.пробел_ли)
      сема = сема.следщ;
    if (сема is null || сема.вид == ТОК.КФ)
      return this;
    return сема;
  }

  /// Находит предшествующую непробельную сему.
  /// Возвращает: 'эту' сему, если предыдущая является ТОК.ГОЛОВА или null.
  Сема* предшНепроб()
  out(сема)
  {
    assert(сема !is null);
  }
  body
  {
    auto сема = предш;
    while (сема !is null && сема.пробел_ли)
      сема = сема.предш;
    if (сема is null || сема.вид == ТОК.ГОЛОВА)
      return this;
    return сема;
  }

  /// Возвращает текстовое определение вида данной семы.
  static ткст вТкст(ТОК вид)
  {
    return семаВТкст[вид];
  }

  /// Adds Флаги.Пробельный в this.флаги.
  проц  установиФлагПробельные()
  {
    this.флаги |= Флаги.Пробельный;
  }

  /// Возвращает да, если из_ сема, внутри которой могут быть символы новой
строки.
  ///
  /// These can be block and nested comments and any ткст literal
  /// except for escape ткст literals.
  бул многострок_ли()
  {
    return вид == ТОК.Ткст && старт[0] != '\\' ||
           вид == ТОК.Комментарий && старт[1] != '/';
  }

  /// Возвращает да, если из_ сема-ключевое слово.
  бул кслово_ли()
  {
    return НачалоКС <= вид && вид <= КонецКС;
  }

  /// Возвращает да, если из_ сема интегрального типа.
  бул интегральныйТип_ли()
  {
    return НачалоИнтегральногоТипа <= вид && вид <= КонецИнтегральногоТипа;
  }

  /// Возвращает да, если из_ сема пробела.
```

```
  бул пробел_ли()
  {
    return !!(флаги & Флаги.Пробельный);
  }

  /// Возвращает да, если из_ a special сема.
  бул спецСема_ли()
  {
    return НачалоСпецСем <= вид && вид <= КонецСпецСем;
  }

version(D2)
{
  /// Возвращает да, если из_ a сема ткст literal.
  бул семаСтроковогоЛитерала_ли()
  {
    return вид == ТОК.Ткст && tok_ткт !is null;
  }
}

  /// Returns да if this сема starts a ДефинцияДекларации.
  бул началоДефДекл_ли()
  {
    return семаНачалаДеклДеф_ли(вид);
  }

  /// Returns да if this сема starts a Инструкция.
  бул началоИнстр_ли()
  {
    return семаНачалаИнстр_ли(вид);
  }

  /// Returns да if this сема starts an ИнструкцияАсм.
  бул началоАсмИнстр_ли()
  {
    return семаНачалаАсмИнстр_ли(вид);
  }

  цел opEquals(ТОК kind2)
  {
    return вид == kind2;
  }

  цел opCmp(Сема* пв)
  {
    return старт < пв.старт;
  }

  /// Возвращает Положение of this сема.
  Положение дайПоложение(бул реальноеПоложение)()
  {
    auto search_t = this.предш;
    // Find предшious нс сема.
    while (search_t.вид != ТОК.Новстр)
      search_t = search_t.предш;
    static if (реальноеПоложение)
    {
      auto путьКФайлу  = search_t.нс.путиКФайлам.исхПуть;
      auto номСтр    = search_t.нс.oriLineNum;
    }
    else
    {
      auto путьКФайлу  = search_t.нс.путиКФайлам.устПуть;
```

```
    auto номСтр   = search_t.нс.oriLineNum - search_t.нс.setLineNum;
  }
  auto началоСтроки = search_t.конец;
  // Determine actual line начало and line число.
  while (1)
  {
    search_t = search_t.следщ;
    if (search_t == this)
      break;
    // Multiline семы must be rescanned for newlines.
    if (search_t.многострок_ли)
    {
      auto p = search_t.старт, конец = search_t.конец;
      while (p != конец)
        if (сканируйНовСтр(p))
        {
          началоСтроки = p;
          ++номСтр;
        }
        else
          ++p;
    }
  }
  return new Положение(путьКФайлу, номСтр, началоСтроки, this.старт);
}

alias дайПоложение!(да) дайРеальноеПоложение;
alias дайПоложение!(нет) дайПоложениеОшибки;

бцел lineCount()
{
  бцел счёт = 1;
  if (this.многострок_ли)
  {
    auto p = this.старт, конец = this.конец;
    while (p != конец)
    {
      if (сканируйНовСтр(p) == '\n')
        ++счёт;
      else
        ++p;
    }
  }
  return счёт;
}

/// Итог the source текст enclosed by the левый and правый сема.
static ткст textSpan(Сема* левый, Сема* правый)
{
  assert(левый.конец <= правый.старт || левый is правый );
  return левый.старт[0 .. правый.конец - левый.старт];
}

/// Uses malloc() в allocate memory for a сема.
new(т_мера размер)
{
  ук p = malloc(размер);
  if (p is null)
    throw new OutOfMemoryException(__FILE__, __LINE__);
  // TODO: Сема.иниц should be all zeros.
  // Maybe use calloc() в avoid this line?
  *cast(Сема*)p = Сема.init;
  return p;
```

```
  }

  /// Deletes a сема using free().
  delete(ук p)
  {
    auto сема = cast(Сема*)p;
    if (сема)
    {
      if(сема.вид == ТОК.HashLine)
        сема.destructHashLineToken();
      else
      {
      version(D2)
        if (сема.семаСтроковогоЛитерала_ли)
          сема.destructTokenStringLiteral();
      }
    }
    free(p);
  }

  проц  destructHashLineToken()
  {
    assert(вид == ТОК.HashLine);
    delete tokLineNum;
    delete tokLineFilespec;
  }

version(D2)
{
  проц  destructTokenStringLiteral()
  {
    assert(вид == ТОК.Ткст);
    assert(старт && *старт == 'q' && старт[1] == '{');
    assert(tok_ткт !is null);
    auto tok_it = tok_ткт;
    auto tok_del = tok_ткт;
    while (tok_it && tok_it.вид != ТОК.КФ)
    {
      tok_it = tok_it.следщ;
      assert(tok_del && tok_del.вид != ТОК.КФ);
      delete tok_del;
      tok_del = tok_it;
    }
  }
}
}

/// Data associated with нс семы.
struct ДанныеНовСтр
{
  struct ФПути
  {
    ткст исхПуть;   /// Original путь в the source текст.
    ткст устПуть;   /// Path установи by #line.
  }
  ФПути* путиКФайлам;
  бцел oriLineNum;  /// Actual line число in the source текст.
  бцел setLineNum;  /// Delta line число установи by #line.
}

/// Returns да if this сема starts a ДефинцияДекларации.
бул семаНачалаДеклДеф_ли(ТОК лекс)
{
```

```
  switch (лекс)
  {
  alias ТОК Т;
  case  Т.Расклад, Т.Прагма, Т.Экспорт, Т.Приватный, Т.Пакет, Т.Защищённый,
        Т.Публичный, Т.Экстерн, Т.Устаревший, Т.Перепись, Т.Абстрактный,
        Т.Синхронизованный, Т.Статический, Т.Окончательный, Т.Конст,
Т.Инвариант/*D 2.0*/,
        Т.Авто, Т.Масштаб, Т.Алиас, Т.Типдеф, Т.Импорт, Т.Перечень, Т.Класс,
        Т.Интерфейс, Т.Структура, Т.Союз, Т.Этот, Т.Тильда, Т.Юниттест,
Т.Отладка,
        Т.Версия, Т.Шаблон, Т.Нов, Т.Удалить, Т.Смесь, Т.ТочкаЗапятая,
        Т.Идентификатор, Т.Точка, Т.Типа:
    return да;
  default:
    if (НачалоИнтегральногоТипа <= лекс && лекс <= КонецИнтегральногоТипа)
      return да;
  }
  return нет;
}

/// Returns да if this сема starts a Инструкция.
бул семаНачалаИнстр_ли(ТОК лекс)
{
  switch (лекс)
  {
  alias ТОК Т;
  case  Т.Расклад, Т.Экстерн, Т.Окончательный, Т.Конст, Т.Авто,
Т.Идентификатор, Т.Точка,
        Т.Типа, Т.Если, Т.Пока, Т.Делай, Т.При, Т.Длявсех, Т.Длявсех_реверс,
        Т.Щит, Т.Реле, Т.Дефолт, Т.Далее, Т.Всё, Т.Итог, Т.Переход,
        Т.Для, Т.Синхронизованный, Т.Пробуй, Т.Брось, Т.Масштаб, Т.Волатайл,
Т.Асм,
        Т.Прагма, Т.Смесь, Т.Статический, Т.Отладка, Т.Версия, Т.Алиас,
Т.ТочкаЗапятая,
        Т.Перечень, Т.Класс, Т.Интерфейс, Т.Структура, Т.Союз, Т.ЛФСкобка,
Т.Типдеф,
        Т.Этот, Т.Супер, Т.Нуль, Т.Истина, Т.Ложь, Т.Цел32, Т.Цел64,
Т.Бцел32,
        Т.Бцел64, Т.Плав32, Т.Плав64, Т.Плав80, Т.Мнимое32,
        Т.Мнимое64, Т.Мнимое80, Т.СимЛитерал, Т.Ткст, Т.ЛКвСкобка,
        Т.Функция, Т.Делегат, Т.Подтвердить, Т.Импорт, Т.Идтипа, Т.Является,
Т.ЛСкобка,
        Т.Трэтс/*D2.0*/, Т.ИБинарное, Т.ПлюсПлюс, Т.МинусМинус, Т.Умножь,
        Т.Минус, Т.Плюс, Т.Не, Т.Тильда, Т.Нов, Т.Удалить, Т.Каст:
    return да;
  default:
    if (НачалоИнтегральногоТипа <= лекс && лекс <= КонецИнтегральногоТипа ||
        НачалоСпецСем <= лекс && лекс <= КонецСпецСем)
      return да;
  }
  return нет;
}

/// Returns да if this сема starts an ИнструкцияАсм.
бул семаНачалаАсмИнстр_ли(ТОК лекс)
{
  switch(лекс)
  {
  alias ТОК Т;
  case Т.Вхо, Т.Цел, Т.Вых, Т.Идентификатор, Т.Расклад, Т.ТочкаЗапятая:
    return да;
  default:
  }
```

```
  return нет;
}
```

module drc.lexer.TokensEnum;

```
import common;

/// Перечисление типов сем.
enum ТОК : бкрат
{
  Неверно,

  Нелегал,
  Комментарий,
  Шебанг,
  HashLine,
  Filespec,
  Новстр,
  Пусто,

  Идентификатор,
  Ткст,
  СимЛитерал,

  // Специальные семы
  ФАЙЛ,
  СТРОКА,
  ДАТА,
  ВРЕМЯ,
  ШТАМПВРЕМЕНИ,
  ПОСТАВЩИК,
  ВЕРСИЯ,

  // Числовые литералы
  Цел32, Цел64, Бцел32, Бцел64,
  // Сканер чисел с плавающей точкой рассчитывает на такой расклад. (ПлавXY +
3 == МнимоеXY)
  Плав32, Плав64, Плав80,
  Мнимое32, Мнимое64, Мнимое80,


  // Скобки
  ЛСкобка,
  ПСкобка,
  ЛКвСкобка,
  ПКвСкобка,
  ЛФСкобка,
  ПФСкобка,

  Точка, Срез, Эллипсис,

  // Операты над числами с плавающей точкой
  Unordered,
  UorE,
  UorG,
  UorGorE,
  UorL,
  UorLorE,
  LorEorG,
  LorG,

  // Нормальные операторы
  Присвоить, Равно, НеРавно, Не,
```

```
  МеньшеРавно, Меньше,
  БольшеРавно, Больше,
  ЛСдвигПрисвой, ЛСдвиг,
  ПСдвигПрисвой,ПСдвиг,
  URShiftAssign, URShift,
  ИлиПрисвой, ИлиЛогическое, ИлиБинарное,
  ИПрисвой, ИЛогическое, ИБинарное,
  ПлюсПрисвой, ПлюсПлюс, Плюс,
  МинусПрисвой, МинусМинус, Минус,
  ДелениеПрисвой, Деление,
  УмножьПрисвой, Умножь,
  МодульПрисвой, Мод,
  ИИлиПрисвой, ИИли,
  CatAssign,
  Тильда,

  Двоеточие,
  ТочкаЗапятая,
  Вопрос,
  Запятая,
  Доллар,

  /* Keywords:
     NB.: Сема.кслово_ли() depends on this список being contiguous.
  */
  Абстрактный, Алиас, Расклад, Асм, Подтвердить, Авто, Тело,
  Всё, Реле, Каст, Кэтч,
  Класс, Конст, Далее,
  Отладка, Дефолт, Делегат, Удалить, Устаревший, Делай,
  Иначе, Перечень, Экспорт, Экстерн, Ложь, Окончательный,
  Finally, При, Длявсех, Длявсех_реверс, Функция, Переход,
  Если, Импорт, Вхо, Вховых,
  Интерфейс, Инвариант, Является, Отложенный, Макрос/+D2.0+/,
  Смесь, Модуль, Нов, Nothrow/+D2.0+/, Нуль, Вых, Перепись, Пакет,
  Прагма, Приватный, Защищённый, Публичный, Pure/+D2.0+/, Реф, Итог,
  Масштаб, Статический, Структура, Супер, Щит, Синхронизованный,
  Шаблон, Этот, Брось, Трэтс/+D2.0+/, Истина, Пробуй, Типдеф, Идтипа,
  Типа, Союз, Юниттест,
  Версия, Волатайл, Пока, Для,
  // Целegral types.
  Сим,   Шим,   Дим, Бул,
  Байт,   Ббайт,   Крат, Бкрат,
  Цел,    Бцел,    Дол,  Бдол,
  Цент,   Бцент,
  Плав,  Дво,  Реал,
  Вплав, Вдво, Вреал,
  Кплав, Кдво, Креал, Проц,

  ГОЛОВА, // старт of linked список
  КФ,
  МАКС
}

alias ТОК.Абстрактный НачалоКС;
alias ТОК.Проц КонецКС;
alias ТОК.Сим НачалоИнтегральногоТипа;
alias ТОК.Проц КонецИнтегральногоТипа;
alias ТОК.ФАЙЛ НачалоСпецСем;
alias ТОК.ВЕРСИЯ КонецСпецСем;

/// Таблица, преобразующая семы каждого вида в текст.
const ткст[ТОК.МАКС] семаВТкст = [
  "Неверный",
```

```
"Нелегал",
"Комментарий",
"#! /shebang/",
"#line",
`"filespec"`,
"НовСтр",
"Пусто",

"Идентификатор",
"Ткст",
"СимЛитерал",

"__FILE__",
"__LINE__",
"__DATE__",
"__TIME__",
"__TIMESTAMP__",
"__VENDOR__",
"__VERSION__",

"Цел32", "Цел64", "Бцел32", "Бцел64",
"Плав32", "Плав64", "Плав80",
"Мнимое32", "Мнимое64", "Мнимое80",

"(",
")",
"[",
"]",
"{",
"}",

".", "..", "...",

"!<>=", // Unordered
"!<>",  // UorE
"!<=",  // UorG
"!<",   // UorGorE
"!>=",  // UorL
"!>",   // UorLorE
"<>=",  // LorEorG
"<>",   // LorG

"=", "==", "!=", "!",
"<=", "<",
">=", ">",
"<<=", "<<",
">>=",">>",
">>>=", ">>>",
"|=", "||", "|",
"&=", "&&", "&",
"+=", "++", "+",
"-=", "--", "-",
"/=", "/",
"*=", "*",
"%=", "%",
"^=", "^",
"~=",
"~",

":",
";",
"?",
```

```
  ",",
  "$",

  "abstract","alias","align","asm","assert","auto","body",
  "break","case","cast","catch",
  "class","const","continue",
  "debug","default","delegate","delete","deprecated","do",
  "else","enum","export","extern","нет","final",
  "finally","for","foreach","foreach_reverse","function","goto",
  "if","import","in","inout",
  "interface","invariant","is","lazy","macro",
  "mixin","module","new","nothrow","null","out","override","package",
  "pragma","private","protected","public","pure","ref","return",
  "scope","static","struct","super","switch","synchronized",
  "template","this","throw","__traits","да","try","typedef","typeid",
  "typeof","union","unittest",
  "version","volatile","while","with",
  // Целегral types.
  "сим",    "шим",    "дим", "бул",
  "байт",    "ббайт",    "крат", "бкрат",
  "цел",     "бцел",     "дол",  "бдол",
  "цент",    "бцент",
  "плав",   "дво",   "реал",
  "вплав",  "вдво",  "вреал",
  "кплав",  "кдво",  "креал", "проц ",

  "ГОЛОВА",
  "КФ"
];
static assert(семаВТкст.length == ТОК.КФ+1);
```

Парсер (parser)

```
module drc.parser.ImportParser;

import drc.parser.Parser;
import drc.ast.Node,
       drc.ast.Declarations,
       drc.ast.Statements;
import drc.SourceText;
import drc.Enums;
import common;

private alias ТОК Т;

/// Облегчённый парсер, который находит лишь инструкции импорта
/// в тексте исходника.
class ПарсерИмпорта : Парсер
{
  this(ИсходныйТекст исхТекст)
  {
    super(исхТекст);
  }

  override СложнаяДекларация старт()
  {
    auto деклы = new СложнаяДекларация;
    super.иниц();
    if (сема.вид == Т.Модуль)
      деклы ~= разборДеклорацииМодуля();
    while (сема.вид != Т.КФ)
      разборДефинициИДекларации(Защита.Нет);
```

```
  return деклы;
}

проц  разборДефинцииБлокаДеклараций(Защита защ)
{
  пропусти(Т.ЛФСкобка);
  while (сема.вид != Т.ПФСкобка && сема.вид != Т.КФ)
    разборДефинцииДекларации(защ);
  пропусти(Т.ПФСкобка);
}

проц  разборБлокаДеклараций(Защита защ)
{
  switch (сема.вид)
  {
  case Т.ЛФСкобка:
    разборДефинцииБлокаДеклараций(защ);
    break;
  case Т.Двоеточие:
    далее();
    while (сема.вид != Т.ПФСкобка && сема.вид != Т.КФ)
      разборДефинцииДекларации(защ);
    break;
  default:
    разборДефинцииДекларации(защ);
  }
}

бул пропускДоЗакрывающего(Т открывающий, Т закрывающий)
{
  alias сема следщ;
  бцел уровень = 1;
  while (1)
  {
    лексер.возьми(следщ);
    if (следщ.вид == открывающий)
      ++уровень;
    else if (следщ.вид == закрывающий && --уровень == 0)
      return да;
    else if (следщ.вид == Т.КФ)
      break;
  }
  return нет;
}

проц  пропускДоСемыПослеЗакрКСкобки()
{
  пропускДоЗакрывающего(Т.ЛСкобка, Т.ПСкобка);
  далее();
}

проц  пропускДоСемыПослеЗакрФСкобки()
{
  пропускДоЗакрывающего(Т.ЛФСкобка, Т.ПФСкобка);
  далее();
}

проц  пропусти(ТОК лекс)
{
  сема.вид == лекс && далее();
}

проц  разборАтрибутаЗащиты()
```

```
{
  Защита защ;
  switch (сема.вид)
  {
  case Т.Приватный:
    защ = Защита.Приватный; break;
  case Т.Пакет:
    защ = Защита.Пакет; break;
  case Т.Защищённый:
    защ = Защита.Защищённый; break;
  case Т.Публичный:
    защ = Защита.Публичный; break;
  case Т.Экспорт:
    защ = Защита.Экспорт; break;
  default:
    assert(0);
  }
  далее();
  разборБлокаДеклараций(защ);
}

проц  разборДефинцииДекларации(Защита защ)
{
  switch (сема.вид)
  {
  case Т.Расклад:
    далее();
    if (сема.вид == Т.ЛСкобка)
      далее(), далее(), далее(); // ( Integer )
    разборБлокаДеклараций(защ);
    break;
  case Т.Прагма:
    далее();
    пропускДоСемыПослеЗакрКСкобки();
    разборБлокаДеклараций(защ);
    break;
  case Т.Экспорт,
       Т.Приватный,
       Т.Пакет,
       Т.Защищённый,
       Т.Публичный:
    разборАтрибутаЗащиты();
    break;
  // Storage classes
  case Т.Экстерн:
    далее();
    сема.вид == Т.ЛСкобка && пропускДоСемыПослеЗакрКСкобки();
    разборБлокаДеклараций(защ);
    break;
  case Т.Конст:
  version(D2)
  {
    if (возьмиСледщ() == Т.ЛСкобка)
      goto случай_Декларация;
  }
  case Т.Перепись,
       Т.Устаревший,
       Т.Абстрактный,
       Т.Синхронизованный,
       // Т.Статический,
       Т.Окончательный,
       Т.Авто,
       Т.Масштаб:
```

```
случай_СтатичАтрибут:
случай_АтрибутИнвариант:
  далее();
  разборБлокаДеклараций(защ);
  break;
// End of storage classes.
case Т.Алиас, Т.Типдеф:
  далее();
  goto случай_Декларация;
case Т.Статический:
  switch (возьмиСледщ())
  {
  case Т.Импорт:
    goto случай_Импорт;
  case Т.Этот:
    далее(), далее(); // static this
    пропускДоСемыПослеЗакрКСкобки();
    разборТелаФункции();
    break;
  case Т.Тильда:
    далее(), далее(), далее(), далее(), далее(); // static ~ this ( )
    разборТелаФункции();
    break;
  case Т.Если:
    далее(), далее();
    пропускДоСемыПослеЗакрКСкобки();
    разборБлокаДеклараций(защ);
    if (сема.вид == Т.Иначе)
      далее(), разборБлокаДеклараций(защ);
    break;
  case Т.Подтвердить:
    далее(), далее(); // static assert
    пропускДоСемыПослеЗакрКСкобки();
    пропусти(Т.ТочкаЗапятая);
    break;
  default:
    goto случай_СтатичАтрибут;
  }
  break;
case Т.Импорт:
случай_Импорт:
  auto декл = разборДекларацииИмпорта();
  декл.установиЗащиту(защ); // Set the защита attribute.
  импорты ~= декл.в!(ДекларацияИмпорта);
  break;
case Т.Перечень:
  далее();
  сема.вид == Т.Идентификатор && далее();
  if (сема.вид == Т.Двоеточие)
  {
    далее();
    while (сема.вид != Т.ЛФСкобка && сема.вид != Т.КФ)
      далее();
  }
  if (сема.вид == Т.ТочкаЗапятая)
    далее();
  else
    пропускДоСемыПослеЗакрФСкобки();
  break;
case Т.Класс:
case Т.Интерфейс:
  далее(), пропусти(Т.Идентификатор); // class Идентификатор
```

```
      сема.вид == Т.ЛСкобка && пропускДоСемыПослеЗакрКСкобки(); // Skip
template парамы.
      if (сема.вид == Т.Двоеточие)
      { // BaseClasses
        далее();
        while (сема.вид != Т.ЛФСкобка && сема.вид != Т.КФ)
          if (сема.вид == Т.ЛСкобка) // Skip ( семы... )
            пропускДоСемыПослеЗакрКСкобки();
          else
            далее();
      }
      if (сема.вид == Т.ТочкаЗапятая)
        далее();
      else
        разборДефинцииБлокаДеклараций(Защита.Нет);
      break;
    case Т.Структура, Т.Союз:
      далее(); пропусти(Т.Идентификатор);
      сема.вид == Т.ЛСкобка && пропускДоСемыПослеЗакрКСкобки();
      if (сема.вид == Т.ТочкаЗапятая)
        далее();
      else
        разборДефинцииБлокаДеклараций(Защита.Нет);
      break;
    case Т.Тильда:
      далее(); // ~
    case Т.Этот:
      далее(); далее(); далее(); // this ( )
      разборТелаФункции();
      break;
    case Т.Инвариант:
    version(D2)
    {
      auto следщ = сема;
      if (возьмиПосле(следщ) == Т.ЛСкобка)
      {
        if (возьмиПосле(следщ) != Т.ПСкобка)
          goto случай_Декларация;
      }
      else
        goto случай_АтрибутИнвариант;
    }
      далее();
      сема.вид == Т.ЛСкобка && пропускДоСемыПослеЗакрКСкобки();
      разборТелаФункции();
      break;
    case Т.Юниттест:
      далее();
      разборТелаФункции();
      break;
    case Т.Отладка:
      далее();
      if (сема.вид == Т.Присвоить)
      {
        далее(), далее(), далее(); // = Condition ;
        break;
      }
      if (сема.вид == Т.ЛСкобка)
        далее(), далее(), далее(); // ( Condition )
      разборБлокаДеклараций(защ);
      if (сема.вид == Т.Иначе)
        далее(), разборБлокаДеклараций(защ);
      break;
```

```
  case Т.Версия:
    далее();
    if (сема.вид == Т.Присвоить)
    {
      далее(), далее(), далее(); // = Condition ;
      break;
    }
    далее(), далее(), далее(); // ( Condition )
    разборБлокаДеклараций(защ);
    if (сема.вид == Т.Иначе)
      далее(), разборБлокаДеклараций(защ);
    break;
  case Т.Шаблон:
    далее();
    пропусти(Т.Идентификатор);
    пропускДоСемыПослеЗакрКСкобки();
    разборДефинцииБлокаДеклараций(Защита.Нет);
    break;
  case Т.Нов:
    далее();
    пропускДоСемыПослеЗакрКСкобки();
    разборТелаФункции();
    break;
  case Т.Удалить:
    далее();
    пропускДоСемыПослеЗакрКСкобки();
    разборТелаФункции();
    break;
  case Т.Смесь:
    while (сема.вид != Т.ТочкаЗапятая && сема.вид != Т.КФ)
      if (сема.вид == Т.ЛСкобка)
        пропускДоСемыПослеЗакрКСкобки();
      else
        далее();
    пропусти(Т.ТочкаЗапятая);
    break;
  case Т.ТочкаЗапятая:
    далее();
    break;
  // Декларация
  case Т.Идентификатор, Т.Точка, Т.Типа:
  случай_Декларация:
    while (сема.вид != Т.ТочкаЗапятая && сема.вид != Т.КФ)
      if (сема.вид == Т.ЛСкобка)
        пропускДоСемыПослеЗакрКСкобки();
      else if (сема.вид == Т.ЛФСкобка)
        пропускДоСемыПослеЗакрФСкобки();
      else
        далее();
    пропусти(Т.ТочкаЗапятая);
    break;
  default:
    if (сема.интегральныйТип_ли)
      goto случай_Декларация;
    далее();
  }
}

ИнструкцияТелаФункции разборТелаФункции()
{
  while (1)
  {
    switch (сема.вид)
```

```
      {
      case Т.ЛФСкобка:
        пропускДоСемыПослеЗакрФСкобки();
        break;
      case Т.ТочкаЗапятая:
        далее();
        break;
      case Т.Вхо:
        далее();
        пропускДоСемыПослеЗакрФСкобки();
        continue;
      case Т.Вых:
        далее();
        if (сема.вид == Т.ЛСкобка)
          далее(), далее(), далее(); // ( Идентификатор )
        пропускДоСемыПослеЗакрФСкобки();
        continue;
      case Т.Тело:
        далее();
        goto case Т.ЛФСкобка;
      default:
      }
      break; // Exit loop.
    }
    return null;
  }
}
```

```
/// Author: Aziz Köksal
/// License: GPL3
/// $(Maturity very high)
module drc.parser.Parser;

import drc.lexer.Lexer,
       drc.lexer.IdTable;
import drc.ast.Node,
       drc.ast.Declarations,
       drc.ast.Statements,
       drc.ast.Expressions,
       drc.ast.Types,
       drc.ast.Parameters;
import drc.Messages;
import drc.Diagnostics;
import drc.Enums;
import drc.CompilerInfo;
import drc.SourceText;
import drc.Unicode;
import common;

import core.Vararg;

/// Парсер производит полный разбор дерева путём исследования
/// списка сем, предоставляемого Лексером.
class Парсер
{
  Лексер лексер; /// Используется для "лексирования" исходного кода.
  Сема* сема; /// Текущая непробельная сема.
  Сема* предыдущСема; /// Предыдущая непробельная сема.

  Диагностика диаг;
  ОшибкаПарсера[] ошибки; /// Массив сообщений об ошибках парсера.

  ДекларацияИмпорта[] импорты; /// ДеклорацииИмпорта в исходном тексте.
```

```
  /// Атрибуты оцениваются на фазе парсирования.
  /// TODO: будет удалено. СемантическаяПроходка1 прозводит обработку
атрибутов.
  ТипКомпоновки типКомпоновки;
  Защита защита; /// определено
  КлассХранения классХранения; /// определено
  бцел размерРаскладки = РАЗМЕР_РАСКЛАДКИ_ПО_УМОЛЧАНИЮ; /// определено

  private alias ТОК Т; /// Часто используется данным классом.
  private alias УзелТипа Тип;

  /// Строит объект Парсер.
  /// Параметры:
  ///   исхТекст = the UTF-8 source код.
  ///   диаг = используется для сбора сообщений об ошибке.
  this(ИсходныйТекст исхТекст, Диагностика диаг = null)
  {
    this.диаг = диаг;
    лексер = new Лексер(исхТекст, диаг);
  }

  /// Переходит к первой семе.
  protected проц  иниц()
  {
    далее();
    предыдущСема = сема;
  }

  /// Переходит к следующей семе.
  проц  далее()
  {
    предыдущСема = сема;
    do
    {
      лексер.следщСема();
      сема = лексер.сема;
    } while (сема.пробел_ли) // Skip whitespace
  }

  /// Запускает парсер и возвращает парсированные декларации.
  СложнаяДекларация старт()
  {
    иниц();
    auto начало = сема;
    auto деклы = new СложнаяДекларация;
    if (сема.вид == Т.Модуль)
      деклы ~= разборДеклорацииМодуля();
    деклы.добавьОпцОтпрыски(разборДефинцииДеклораций());
    установи(деклы, начало);
    return деклы;
  }

  /// Запускает парсер и возвращает парсированные выражения.
  Выражение старт2()
  {
    иниц();
    return разборВыражения();
  }

  // Members related в the method пробуй_().
  бцел пробуем; /// Больше than 0 if Парсер is in пробуй_().
  бцел счётОшибок; /// Используется для отслеживания числа ошибок при обороте
пробуй_().
```

```
  /// Этот method executes the delegate методРазбора and when an ошибка
occurred
  /// the state of the лексер and парсер is restored.
  /// Возвращает: the return значение of методРазбора().
  ТипИтога пробуй_(ТипИтога)(ТипИтога delegate() методРазбора, out бул успех)
  {
    // Save члены.
    auto старСема     = this.сема;
    auto старПредшСема = this.предыдущСема;
    auto старСчёт     = this.счётОшибок;

    ++пробуем;
    auto результат = методРазбора();
    --пробуем;
    // Check if an ошибка occurred.
    if (счётОшибок != старСчёт)
    { // Restore члены.
      сема       = старСема;
      предыдущСема   = старПредшСема;
      лексер.сема = старСема;
      счётОшибок  = старСчёт;
      успех = нет;
    }
    else
      успех = да;
    return результат;
  }

  /// Вызывает неудачное завершение текущего вызова пробуй_().
  проц  провал_пробы()
  {
    assert(пробуем);
    счётОшибок++;
  }

  /// Устанавливает начало и конец семы узла синтактического древа.
  Класс установи(Класс)(Класс узел, Сема* начало)
  {
    узел.установиСемы(начало, this.предыдущСема);
    return узел;
  }

  ///Устанавливает начало и конец семы узла синтактического древа.
  Класс установи(Класс)(Класс узел, Сема* начало, Сема* конец)
  {
    узел.установиСемы(начало, конец);
    return узел;
  }

  /// Returns да if установи() has been called on a узел.
  static бул узелУстановлен(Узел узел)
  {
    return узел.начало !is null && узел.конец !is null;
  }

  /// Возвращает вид следующей семы.
  ТОК возьмиСледщ()
  {
    Сема* следщ = сема;
    do
      лексер.возьми(следщ);
    while (следщ.пробел_ли) // Skip whitespace
```

```
    return следщ.вид;
  }

  /// Возвращает род семы, следующей за t.
  ТОК возьмиПосле(ref Сема* t)
  {
    assert(t !is null);
    do
      лексер.возьми(t);
    while (t.пробел_ли) // Skip whitespace
    return t.вид;
  }

  /// Проверяет теущую сему на соответствие k по виду и возвращает да.
  бул проверено()(ТОК k) // Templatized, so it's inlined.
  {
    return сема.вид == k ? (далее(), да) : нет;
  }

  /// Проверяет, чтобы текущая сема была ожидаемого вида,
  /// затем приступает за следующую сему.
  проц  пропусти()(ТОК ожидаемыйВид)
  {
    assert(сема.вид == ожидаемыйВид /+|| *(цел*).иниц+/, сема.исхТекст());
    далее();
  }


/+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  |                        Методы парсинга деклараций
|

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+/

  Декларация разборДеклорацииМодуля()
  {
    auto начало = сема;
    пропусти(Т.Модуль);
    ПКИМодуля пкиМодуля;
    do
      пкиМодуля ~= требуетсяИдентификатор(сооб.ОжидалсяИдентификаторМодуля);
    while (проверено(Т.Точка))
    требуется(Т.ТочкаЗапятая);
    return установи(new ДекларацияМодуля(пкиМодуля), начало);
  }

  /// Парсирует "Дефиниции Деклараций" (определения объявлений) до конца
файла.
  /// $(PRE
  /// DeclDefs :=
  ///     DeclDef
  ///     DeclDefs
  /// )
  Декларация[] разборДефинцииДеклараций()
  {
    Декларация[] деклы;
    while (сема.вид != Т.КФ)
      деклы ~= разборДефинцииДекларации();
    return деклы;
  }

  /// Парсирует тело шаблона, класса, интерфейса структуры или союза.
  /// $(PRE
```

```
/// DeclDefsBlock :=
///     { }
///     { DeclDefs }
/// )
СложнаяДекларация разборТелаДефинициидекларации()
{
  // Save attributes.
  auto типКомпоновки  = this.типКомпоновки;
  auto защита    = this.защита;
  auto классХранения = this.классХранения;
  // Clear attributes.
  this.типКомпоновки  = ТипКомпоновки.Нет;
  this.защита    = Защита.Нет;
  this.классХранения = КлассХранения.Нет;

  // Parse body.
  auto начало = сема;
  auto деклы = new СложнаяДекларация;
  требуется(Т.ЛФСкобка);
  while (сема.вид != Т.ПФСкобка && сема.вид != Т.КФ)
    деклы ~= разборДефинициидекларации();
  требуетсяЗакрыв(Т.ПФСкобка, начало);
  установи(деклы, начало);

  // Restore original значения.
  this.типКомпоновки  = типКомпоновки;
  this.защита    = защита;
  this.классХранения = классХранения;

  return деклы;
}

/// Парсирует ДефинцияДекларации.
Декларация разборДефинициидекларации()
out(декл)
{ assert(узелУстановлен(декл)); }
body
{
  auto начало = сема;
  Декларация декл;
  switch (сема.вид)
  {
  case Т.Расклад,
       Т.Прагма,
       // Защита attributes
       Т.Экспорт,
       Т.Приватный,
       Т.Пакет,
       Т.Защищённый,
       Т.Публичный:
    декл = разборИдентификатораАтрибута();
    break;
  // Storage classes
  case Т.Экстерн,
       Т.Устаревший,
       Т.Перепись,
       Т.Абстрактный,
       Т.Синхронизованный,
       //Т.Статический,
       Т.Окончательный,
       Т.Конст,
       //Т.Инвариант, // D 2.0
       Т.Авто,
```

```
     Т.Масштаб:
случай_СтатичАтрибут:
случай_АтрибутИнвариант: // D 2.0
случай_АтрибутПеречень: // D 2.0
  return разборАтрибутаСохранения();
case Т.Алиас:
  далее();
  декл = new ДекларацияАлиаса(разборПеременнойИлиФункции());
  break;
case Т.Типдеф:
  далее();
  декл = new ДекларацияТипдефа(разборПеременнойИлиФункции());
  break;
case Т.Статический:
  switch (возьмиСледщ())
  {
  case Т.Импорт:
    goto случай_Импорт;
  case Т.Этот:
    декл = разборДекларацииСтатичКонструктора();
    break;
  case Т.Тильда:
    декл = разборДекларацииСтатичДеструктора();
    break;
  case Т.Если:
    декл = парсируйДекларациюСтатичЕсли();
    break;
  case Т.Подтвердить:
    декл = парсируйДекларациюСтатичАссерта();
    break;
  default:
    goto случай_СтатичАтрибут;
  }
  break;
case Т.Импорт:
случай_Импорт:
  декл = разборДекларацииИмпорта();
  импорты ~= декл.в!(ДекларацияИмпорта);
  // Handle specially. КлассХранения mustn't be установи.
  декл.установиЗащиту(this.защита);
  return установи(декл, начало);
case Т.Перечень:
version(D2)
{
  if (манифестПеречня_ли())
    goto случай_АтрибутПеречень;
}
  декл = разборДекларацииПеречня();
  break;
case Т.Класс:
  декл = разборДекларацииКласса();
  break;
case Т.Интерфейс:
  декл = разборДекларацииИнтерфейса();
  break;
case Т.Структура, Т.Союз:
  декл = разборДекларацииСтруктурыИлиСоюза();
  break;
case Т.Этот:
  декл = разборДекларацииКонструктора();
  break;
case Т.Тильда:
  декл = разборДекларацииДеструктора();
```

```
      break;
    case Т.Инвариант:
    version(D2)
    {
      auto следщ = сема;
      if (возьмиПосле(следщ) == Т.ЛСкобка)
      {
        if (возьмиПосле(следщ) != Т.ПСкобка)
          goto случай_Декларация;  // invariant ( Тип )
      }
      else
        goto случай_АтрибутИнвариант; // invariant as КлассХранения.
    }
      декл = разборДеклорацииИнварианта(); // invariant ( )
      break;
    case Т.Юниттест:
      декл = разборДеклорацииЮниттеста();
      break;
    case Т.Отладка:
      декл = разборДеклорацииОтладки();
      break;
    case Т.Версия:
      декл = парсируйДекларациюВерсии();
      break;
    case Т.Шаблон:
      декл = парсируйДекларациюШаблона();
      break;
    case Т.Нов:
      декл = парсируйДекларациюНов();
      break;
    case Т.Удалить:
      декл = парсируйДекларациюУдалить();
      break;
    case Т.Смесь:
      декл = парсируйМиксин!(ДекларацияСмеси)();
      break;
    case Т.ТочкаЗапятая:
      далее();
      декл = new ПустаяДекларация();
      break;
    // Декларация
    case Т.Идентификатор, Т.Точка, Т.Типа:
    случай_Декларация:
      return разборПеременнойИлиФункции(this.классХранения, this.защита,
this.типКомпоновки);
    default:
      if (сема.интегральныйТип_ли)
        goto случай_Декларация;
      else if (сема.вид == Т.Модуль)
      {
        декл = разборДеклорацииМодуля();
        ошибка(начало, сооб.ДеклорацияМодуляНеПервая);
        return декл;
      }

      декл = new НелегальнаяДекларация();
      // Skip в следщ valid сема.
      do
        далее();
      while (!сема.началоДефДекл_ли &&
              сема.вид != Т.ПФСкобка &&
              сема.вид != Т.КФ)
      auto текст = Сема.textSpan(начало, this.предыдущСема);
```

```
      ошибка(начало, сооб.НелегальнаяДекларация, текст);
    }
    декл.установиЗащиту(this.защита);
    декл.установиКлассХранения(this.классХранения);
    assert(!узелУстановлен(декл));
    установи(декл, начало);
    return декл;
  }

  /// Parses a DeclarationsBlock.
  /// $(PRE
  /// DeclarationsBlock :=
  ///     : DeclDefs
  ///     { }
  ///     { DeclDefs }
  ///     DeclDef
  /// )
  Декларация разборБлокаДеклараций(/+бул noДвоеточие = нет+/)
  {
    Декларация d;
    switch (сема.вид)
    {
    case Т.ЛФСкобка:
      auto начало = сема;
      далее();
      auto деклы = new СложнаяДекларация;
      while (сема.вид != Т.ПФСкобка && сема.вид != Т.КФ)
        деклы ~= разборДефинициИДекларации();
      требуетсяЗакрыв(Т.ПФСкобка, начало);
      d = установи(деклы, начало);
      break;
    case Т.Двоеточие:
      // if (noДвоеточие == да)
      //   goto default;
      далее();
      auto начало = сема;
      auto деклы = new СложнаяДекларация;
      while (сема.вид != Т.ПФСкобка && сема.вид != Т.КФ)
        деклы ~= разборДефинициИДекларации();
      d = установи(деклы, начало);
      break;
    default:
      d = разборДефинициИДекларации();
    }
    assert(узелУстановлен(d));
    return d;
  }

  // Декларация разборБлокаДеклараційNoДвоеточие()
  // {
  //   return разборБлокаДеклараций(да);
  // }

  /// Parses either a ДекларацияПеременной or a ДекларацияФункции.
  /// Параметры:
  ///   кхр = предшiously parsed storage classes
  ///   защита = предшiously parsed защита attribute
  ///   типКомпоновки = предшiously parsed linkage тип
  ///   testAutoDeclaration = whether в check for an ДекларацияАвто
  ///   optionalParameterList = a hint for how в разбор C-style function
pointers
  Декларация разборПеременнойИлиФункции(КлассХранения кхр =
КлассХранения.Нет,
```

```
                                        Защита защита = Защита.Нет,
                                        ТипКомпоновки типКомпоновки =
ТипКомпоновки.Нет,
                                        бул testAutoDeclaration = нет,
                                        бул optionalParameterList = да)
  {
    auto начало = сема;
    Тип тип;
    Идентификатор* имя;

    // Check for ДекларацияАвто: КлассыСохранения Идентификатор =
    if (testAutoDeclaration && сема.вид == Т.Идентификатор)
    {
      auto вид = возьмиСледщ();
      if (вид == Т.Присвоить)
      { // Авто переменная declaration.
        имя = сема.идент;
        пропусти(Т.Идентификатор);
        goto LparseVariables;
      }
      else version(D2) if (вид == Т.ЛСкобка)
      { // Check for auto return тип template function.
        // КлассыСохранения Name ( TemplateParameterList ) ( ParameterList )
        имя = сема.идент;
        auto следщ = сема;
        возьмиПосле(следщ);
        if (семаПослеСкобкиЯвляется(Т.ЛСкобка, следщ))
        {
          пропусти(Т.Идентификатор);
          assert(сема.вид == Т.ЛСкобка);
          goto LparseTPList; // Далее with parsing a template function.
        }
      }
    }

    тип = разборТипа(); // VariableType or ТипИтога

    if (сема.вид == Т.ЛСкобка)
    { // Указатели на функции в стиле Си усложняют грамматику.
      // С ними приходится иметь дело отдельно, в масштабе функции.
      // Пример:
      //   проц  foo() {
      //     // Указатель на функцию, принимающий целое число и возвращающий
'some_type'.
      //     some_type (*p_func)(цел);
      //     // Вхо the following case precedence is given в a
ВыражениеВызов.
      //     something(*p); // 'something' may be a function/method or an
объект having opCall overloaded.
      //   }
      //   // A pointer в a function taking no параметры and returning
'something'.
      //   something(*p);
      тип = разборТипаУказательНаФункциюСи(тип, имя, optionalParameterList);
    }
    else if (возьмиСледщ() == Т.ЛСкобка)
    { // Тип FunctionName ( ParameterList ) FunctionBody
      имя = требуетсяИдентификатор(сооб.ОжидалосьНазваниеФункции);
      имя || далее(); // Skip non-identifier сема.
      assert(сема.вид == Т.ЛСкобка);
      // It's a function declaration
      ПараметрыШаблона шпарамы;
      Выражение констрейнт;
```

```
      if (семаПослеСкобкиЯвляется(Т.ЛСкобка))
      LparseTPList:
        // ( TemplateParameterList ) ( ParameterList )
        шпарамы = разборСпискаПараметровШаблона();

      auto парамы = разборСпискаПараметров();
    version(D2)
    {
      if (шпарамы) // Если ( ConstraintExpression )
        констрейнт = разборДополнительногоКонстрейнта();
      switch (сема.вид)
      {
      case Т.Конст:
        кхр |= КлассХранения.Конст;
        далее();
        break;
      case Т.Инвариант:
        кхр |= КлассХранения.Инвариант;
        далее();
        break;
      default:
      }
    }
      // ТипИтога FunctionName ( ParameterList )
      auto телоФунк = разборТелаФункции();
      auto дф = new ДекларацияФункции(тип, имя,/+ шпарамы,+/ парамы,
телоФунк);
      дф.установиКлассХранения(кхр);
      дф.установиТипКомпоновки(типКомпоновки);
      дф.установиЗащиту(защита);
      if (шпарамы)
      {
        auto d = поместиДекларациюВнутреннегоШаблона(начало, имя, дф,
шпарамы, констрейнт);
        d.установиКлассХранения(кхр);
        d.установиЗащиту(защита);
        return установи(d, начало);
      }
      return установи(дф, начало);
    }
    else
    { // Тип VariableName DeclaratorSuffix
      имя = требуетсяИдентификатор(сооб.ОжидалосьНазваниеПеременной);
      тип = разборСуффиксаДекларатора(тип);
    }

  LparseVariables:
    // It's a переменные declaration.
    Идентификатор*[] имена = [имя]; // One identifier has been parsed
already.
    Выражение[] значения;
    goto LenterLoop; // Enter the loop and check for an initializer.
    while (проверено(Т.Запятая))
    {
      имена ~= требуетсяИдентификатор(сооб.ОжидалосьНазваниеПеременной);
    LenterLoop:
      if (проверено(Т.Присвоить))
        значения ~= разборИнициализатора();
      else
        значения ~= null;
    }
    требуется(Т.ТочкаЗапятая);
```

```
  auto d = new ДекларацияПеременных(тип, имена, значения);
  d.установиКлассХранения(кхр);
  d.установиТипКомпоновки(типКомпоновки);
  d.установиЗащиту(защита);
  return установи(d, начало);
}

/// Parses a переменная initializer.
Выражение разборИнициализатора()
{
  if (сема.вид == Т.Проц)
  {
    auto начало = сема;
    auto следщ = возьмиСледщ();
    if (следщ == Т.Запятая || следщ == Т.ТочкаЗапятая)
    {
      пропусти(Т.Проц);
      return установи(new ВыражениеИницПроц(), начало);
    }
  }
  return разборНеПроцИнициализатора();
}

Выражение разборНеПроцИнициализатора()
{
  auto начало = сема;
  Выражение иниц;
  switch (сема.вид)
  {
  case Т.ЛКвСкобка:
    // ArrayInitializer:
    //          [ ]
    //          [ ArrayMemberInitializations ]
    Выражение[] ключи;
    Выражение[] значения;

    пропусти(Т.ЛКвСкобка);
    while (сема.вид != Т.ПКвСкобка)
    {
      auto в = разборНеПроцИнициализатора();
      if (проверено(Т.Двоеточие))
      {
        ключи ~= в;
        значения ~= разборНеПроцИнициализатора();
      }
      else
      {
        ключи ~= null;
        значения ~= в;
      }

      if (!проверено(Т.Запятая))
        break;
    }
    требуетсяЗакрыв(Т.ПКвСкобка, начало);
    иниц = new ВыражениеИницМассива(ключи, значения);
    break;
  case Т.ЛФСкобка:
    // StructInitializer:
    //          { }
    //          { StructMemberInitializers }
    Выражение разборИнициализатораСтрукт()
    {
```

```
      Идентификатор*[] иденты;
      Выражение[] значения;

      пропусти(Т.ЛФСкобка);
      while (сема.вид != Т.ПФСкобка)
      {
        if (сема.вид == Т.Идентификатор &&
            // Peek for colon в see if this is a член identifier.
            возьмиСледщ() == Т.Двоеточие)
        {
          иденты ~= сема.идент;
          пропусти(Т.Идентификатор), пропусти(Т.Двоеточие);
        }
        else
          иденты ~= null;

        // NonVoidInitializer
        значения ~= разборНеПроцИнициализатора();

        if (!проверено(Т.Запятая))
          break;
      }
      требуетсяЗакрыв(Т.ПФСкобка, начало);
      return new ВыражениеИницСтруктуры(иденты, значения);
    }

    бул успех;
    auto si = пробуй_(&разборИнициализатораСтрукт, успех);
    if (успех)
    {
      иниц = si;
      break;
    }
    assert(сема.вид == Т.ЛФСкобка);
    //goto default;
  default:
    иниц = разборВыраженияПрисвой();
  }
  установи(иниц, начало);
  return иниц;
}

ИнструкцияТелаФункции разборТелаФункции()
{
  auto начало = сема;
  auto func = new ИнструкцияТелаФункции;
  while (1)
  {
    switch (сема.вид)
    {
    case Т.ЛФСкобка:
      func.телоФунк = разборИнструкций();
      break;
    case Т.ТочкаЗапятая:
      далее();
      break;
    case Т.Вхо:
      if (func.телоВхо)
        ошибка(ИДС.КонтрактИн);
      далее();
      func.телоВхо = разборИнструкций();
      continue;
    case Т.Вых:
```

```
      if (func.телоВых)
        ошибка(ИДС.КонтрактАут);
      далее();
      if (проверено(Т.ЛСкобка))
      {
        auto leftParen = this.предыдущСема;
        func.outIdent = требуетсяИдентификатор(сооб.ОжидалсяИдентификатор);
        требуетсяЗакрыв(Т.ПСкобка, leftParen);
      }
      func.телоВых = разборИнструкций();
      continue;
    case Т.Тело:
      далее();
      goto case Т.ЛФСкобка;
    default:
      ошибка2(сооб.ОжидалосьТелоФункции, сема);
    }
    break; // Exit loop.
  }
  установи(func, начало);
  func.завершиКонструкцию();
  return func;
}

ТипКомпоновки разборТипаКомпоновки()
{
  ТипКомпоновки типКомпоновки;

  if (!проверено(Т.ЛСкобка))
    return типКомпоновки;

  if (проверено(Т.ПСкобка))
  { // extern()
    ошибка(ИДС.ОтсутствуетТипКомпоновки);
    return типКомпоновки;
  }

  auto identTok = требуетсяИд();

  ВИД видИд = identTok ? identTok.идент.видИд : ВИД.Нуль;

  switch (видИд)
  {
  case ВИД.C:
    if (проверено(Т.ПлюсПлюс))
    {
      типКомпоновки = ТипКомпоновки.Cpp;
      break;
    }
    типКомпоновки = ТипКомпоновки.C;
    break;
  case ВИД.D:
    типКомпоновки = ТипКомпоновки.D;
    break;
  case ВИД.Windows:
    типКомпоновки = ТипКомпоновки.Windows;
    break;
  case ВИД.Pascal:
    типКомпоновки = ТипКомпоновки.Pascal;
    break;
  case ВИД.System:
    типКомпоновки = ТипКомпоновки.Система;
    break;
```

```
    default:
      ошибка2(ИДС.НеопознанныйТипКомпоновки, сема);
    }
    требуется(Т.ПСкобка);
    return типКомпоновки;
  }

  проц  проверьТипКомпоновки(ref ТипКомпоновки предш_тк, ТипКомпоновки тк,
Сема* начало)
  {
    if (предш_тк == ТипКомпоновки.Нет)
      предш_тк = тк;
    else
      ошибка(начало, сооб.ПовторяющийсяТипЛинковки, Сема.textSpan(начало,
this.предыдущСема));
  }

  Декларация разборАтрибутаСохранения()
  {
    КлассХранения кхр, stc_tmp;
    ТипКомпоновки предш_типКомпоновки;

    auto saved_storageClass = this.классХранения; // Save.
    // Nested function.
    Декларация разбор()
    {
      Декларация декл;
      auto начало = сема;
      switch (сема.вид)
      {
      case Т.Экстерн:
        if (возьмиСледщ() != Т.ЛСкобка)
        {
          stc_tmp = КлассХранения.Экстерн;
          goto Lcommon;
        }

        далее();
        auto типКомпоновки = разборТипаКомпоновки();
        проверьТипКомпоновки(предш_типКомпоновки, типКомпоновки, начало);

        auto saved = this.типКомпоновки; // Save.
        this.типКомпоновки = типКомпоновки; // Set.
        декл = new ДекларацияКомпоновки(типКомпоновки, разбор());
        установи(декл, начало);
        this.типКомпоновки = saved; // Restore.
        break;
      case Т.Перепись:
        stc_tmp = КлассХранения.Перепись;
        goto Lcommon;
      case Т.Устаревший:
        stc_tmp = КлассХранения.Устаревший;
        goto Lcommon;
      case Т.Абстрактный:
        stc_tmp = КлассХранения.Абстрактный;
        goto Lcommon;
      case Т.Синхронизованный:
        stc_tmp = КлассХранения.Синхронизованный;
        goto Lcommon;
      case Т.Статический:
        stc_tmp = КлассХранения.Статический;
        goto Lcommon;
      case Т.Окончательный:
```

```
  stc_tmp = КлассХранения.Окончательный;
  goto Lcommon;
case Т.Конст:
version(D2)
{
  if (возьмиСледщ() == Т.ЛСкобка)
    goto случай_Декларация;
}
  stc_tmp = КлассХранения.Конст;
  goto Lcommon;
version(D2)
{
case Т.Инвариант: // D 2.0
  auto следщ = сема;
  if (возьмиПосле(следщ) == Т.ЛСкобка)
  {
    if (возьмиПосле(следщ) != Т.ПСкобка)
      goto случай_Декларация; // invariant ( Тип )
    декл = разборДекларацииИнварианта(); // invariant ( )
    // NB: this must be similar в the код at the конец of
    //     разборДефинцииДекларации().
    декл.установиЗащиту(this.защита);
    декл.установиКлассХранения(кхр);
    установи(декл, начало);
    break;
  }
  // инвариант как классХранения.
  stc_tmp = КлассХранения.Инвариант;
  goto Lcommon;
case Т.Перечень: // D 2.0
  if (!манифестПеречня_ли())
  { // A normal enum declaration.
    декл = разборДекларацииПеречня();
    // NB: this must be similar в the код at the конец of
    //     разборДефинцииДекларации().
    декл.установиЗащиту(this.защита);
    декл.установиКлассХранения(кхр);
    установи(декл, начало);
    break;
  }
  // enum as КлассХранения.
  stc_tmp = КлассХранения.Манифест;
  goto Lcommon;
} // version(D2)
case Т.Авто:
  stc_tmp = КлассХранения.Авто;
  goto Lcommon;
case Т.Масштаб:
  stc_tmp = КлассХранения.Масштаб;
  goto Lcommon;
Lcommon:
  // Issue ошибка if redundant.
  if (кхр & stc_tmp)
    ошибка2(ИДС.ПовторяющийсяКлассХранения, сема);
  else
    кхр |= stc_tmp;

  далее();
  декл = new ДекларацияКлассаХранения(stc_tmp, разбор());
  установи(декл, начало);
  break;
case Т.Идентификатор:
случай_Декларация:
```

```
        // Этот could be a normal Декларация or an ДекларацияАвто
        декл = разборПеременнойИлиФункции(кхр, this.защита,
предш_типКомпоновки, да);
        break;
      default:
        this.классХранения = кхр; // Set.
        декл = разборБлокаДеклараций();
        this.классХранения = saved_storageClass; // Reset.
      }
      assert(узелУстановлен(декл));
      return декл;
    }
    return разбор();
  }

  бцел разборАтрибутаАлайн()
  {
    пропусти(Т.Расклад);
    бцел размер = РАЗМЕР_РАСКЛАДКИ_ПО_УМОЛЧАНИЮ; // Global default.
    if (проверено(Т.ЛСкобка))
    {
      if (сема.вид == Т.Цел32)
        (размер = сема.цел_), пропусти(Т.Цел32);
      else
        ожидаемое(Т.Цел32);
      требуется(Т.ПСкобка);
    }
    return размер;
  }

  Декларация разборИдентификатораАтрибута()
  {
    Декларация декл;

    switch (сема.вид)
    {
    case Т.Расклад:
      бцел размерРаскладки = разборАтрибутаАлайн();
      auto saved = this.размерРаскладки; // Save.
      this.размерРаскладки = размерРаскладки; // Set.
      декл = new ДекларацияРазложи(размерРаскладки, разборБлокаДеклараций());
      this.размерРаскладки = saved; // Restore.
      break;
    case Т.Прагма:
      // Прагма:
      //     pragma ( Идентификатор )
      //     pragma ( Идентификатор , ExpressionList )
      далее();
      Идентификатор* идент;
      Выражение[] арги;

      требуется(Т.ЛСкобка);
      идент = требуетсяИдентификатор(сооб.ОжидалсяИдентификаторПрагмы);

      if (проверено(Т.Запятая))
        арги = разборСпискаВыражений();
      требуется(Т.ПСкобка);

      декл = new ДекларацияПрагмы(идент, арги, разборБлокаДеклараций());
      break;
    default:
      // Защита attributes
      Защита защ;
```

```
      switch (сема.вид)
      {
      case Т.Приватный:
        защ = Защита.Приватный; break;
      case Т.Пакет:
        защ = Защита.Пакет; break;
      case Т.Защищённый:
        защ = Защита.Защищённый; break;
      case Т.Публичный:
        защ = Защита.Публичный; break;
      case Т.Экспорт:
        защ = Защита.Экспорт; break;
      default:
        assert(0);
      }
      далее();
      auto saved = this.защита; // Save.
      this.защита = защ; // Set.
      декл = new ДекларацияЗащиты(защ, разборБлокаДеклараций());
      this.защита = saved; // Restore.
    }
    return декл;
  }

  Декларация разборДекларацииИмпорта()
  {
    бул статический_ли = проверено(Т.Статический);
    пропусти(Т.Импорт);

    ПКИМодуля[] пкиМодулей;
    Идентификатор*[] алиасыМодуля;
    Идентификатор*[] связанныеИмена;
    Идентификатор*[] связанныеАлиасы;

    do
    {
      ПКИМодуля пкиМодуля;
      Идентификатор* moduleAlias;
      // AliasName = ModuleName
      if (возьмиСледщ() == Т.Присвоить)
      {
        moduleAlias = требуетсяИдентификатор(сооб.ExpectedAliasModuleName);
        пропусти(Т.Присвоить);
      }
      // Идентификатор ("." Идентификатор)*
      do
        пкиМодуля ~=
требуетсяИдентификатор(сооб.ОжидалсяИдентификаторМодуля);
      while (проверено(Т.Точка))
      // Push identifiers.
      пкиМодулей ~= пкиМодуля;
      алиасыМодуля ~= moduleAlias;
    } while (проверено(Т.Запятая))

    if (проверено(Т.Двоеточие))
    { // BindAlias "=" BindName ("," BindAlias "=" BindName)*;
      // BindName ("," BindName)*;
      do
      {
        Идентификатор* bindAlias;
        // BindAlias = BindName
        if (возьмиСледщ() == Т.Присвоить)
        {
```

```
          bindAlias = требуетсяИдентификатор(сооб.ОжидаемоеИмяАлиасаИмпорта);
          пропусти(Т.Присвоить);
        }
        // Push identifiers.
        связанныеИмена ~= требуетсяИдентификатор(сооб.ОжидаемоеИмяИмпорта);
        связанныеАлиасы ~= bindAlias;
      } while (проверено(Т.Запятая))
    }
    требуется(Т.ТочкаЗапятая);

    return new ДекларацияИмпорта(пкиМодулей, алиасыМодуля, связанныеИмена,
связанныеАлиасы, статический_ли);
  }

version(D2)
{
  /// Возвращает да, если из_ an enum manifest or
  /// нет if it's a normal enum declaration.
  бул манифестПеречня_ли()
  {
    assert(сема.вид == Т.Перечень);
    auto следщ = сема;
    auto вид = возьмиПосле(следщ);
    if (вид == Т.Двоеточие || вид == Т.ЛФСкобка)
      return нет; // Anonymous enum.
    else if (вид == Т.Идентификатор)
    {
      вид = возьмиПосле(следщ);
      if (вид == Т.Двоеточие || вид == Т.ЛФСкобка || вид == Т.ТочкаЗапятая)
        return нет; // Named enum.
    }
    return да; // Манифест enum.
  }
}

  Декларация разборДеклорацииПеречня()
  {
    пропусти(Т.Перечень);

    Идентификатор* имяПеречня;
    Тип типОснова;
    ДекларацияЧленаПеречня[] члены;
    бул естьТело;

    имяПеречня = дополнительныйИдентификатор();

    if (проверено(Т.Двоеточие))
      типОснова = разборБазовогоТипа();

    if (имяПеречня && проверено(Т.ТочкаЗапятая))
    {}
    else if (проверено(Т.ЛФСкобка))
    {
      auto леваяФСкобка = this.предыдущСема;
      естьТело = да;
      while (сема.вид != Т.ПФСкобка)
      {
        auto начало = сема;

        Тип тип;
      version(D2)
      {
        бул успех;
```

```
        пробуй_({
          // Тип Идентификатор = ВыражениеПрисвой
          тип = разборТипа(); // Set outer тип переменная.
          if (сема.вид != Т.Идентификатор)
            провал_пробы(), (тип = null);
          return null;
        }, успех);
      }

        auto имя = требуетсяИдентификатор(сооб.ОжидалсяЧленПеречня);
        Выражение значение;

        if (проверено(Т.Присвоить))
          значение = разборВыраженияПрисвой();

        члены ~= установи(new ДекларацияЧленаПеречня(тип, имя, значение),
начало);

        if (!проверено(Т.Запятая))
          break;
      }
      требуетсяЗакрыв(Т.ПФСкобка, леваяФСкобка);
    }
    else
      ошибка2(сооб.ОжидалосьТелоПеречня, сема);

    return new ДекларацияПеречня(имяПеречня, типОснова, члены, естьТело);
  }

  /// Wraps a declaration внутри a template declaration.
  /// Параметры:
  ///   начало = начало сема of декл.
  ///   имя = имя of декл.
  ///   декл = the declaration в be wrapped.
  ///   шпарамы = the template параметры.
  ///   констрейнт = the констрейнт выражение.
  ДекларацияШаблона поместиДекларациюВнутреннегоШаблона(Сема* начало,
                                                        Идентификатор* имя,
                                                        Декларация декл,
                                                        ПараметрыШаблона шпарамы,
                                                        Выражение констрейнт)
  {
    установи(декл, начало);
    auto cd = new СложнаяДекларация;
    cd ~= декл;
    установи(cd, начало);
    return new ДекларацияШаблона(имя, шпарамы, констрейнт, cd);
  }

  Декларация разборДекларацииКласса()
  {
    auto начало = сема;
    пропусти(Т.Класс);

    Идентификатор* имяКласса;
    ПараметрыШаблона шпарамы;
    Выражение констрейнт;
    ТипКлассОснова[] основы;
    СложнаяДекларация деклы;

    имяКласса = требуетсяИдентификатор(сооб.ОжидалосьНазваниеКласса);

    if (сема.вид == Т.ЛСкобка)
```

```
    {
      шпарамы = разборСпискаПараметровШаблона();
      version(D2) констрейнт = разборДополнительногоКонстрейнта();
    }

    if (сема.вид == Т.Двоеточие)
      основы = разборБазовыхКлассов();

    if (основы.length == 0 && проверено(Т.ТочкаЗапятая))
    {}
    else if (сема.вид == Т.ЛФСкобка)
      деклы = разборТелаДефинициидекларации();
    else
      ошибка2(сооб.ОжидалосьТелоКласса, сема);

    Декларация d = new ДекларацияКласса(имяКласса, /+шпарамы, +/основы,
деклы);
    if (шпарамы)
      d = поместиДекларациюВнутреннегоШаблона(начало, имяКласса, d, шпарамы,
констрейнт);
    return d;
  }

  ТипКлассОснова[] разборБазовыхКлассов(бул colonLeadsOff = да)
  {
    colonLeadsOff && пропусти(Т.Двоеточие);

    ТипКлассОснова[] основы;
    do
    {
      Защита защ = Защита.Публичный;
      switch (сема.вид)
      {
      case Т.Идентификатор, Т.Точка, Т.Типа: goto LparseBasicType;
      case Т.Приватный:   защ = Защита.Приватный;   break;
      case Т.Защищённый: защ = Защита.Защищённый; break;
      case Т.Пакет:   защ = Защита.Пакет;   break;
      case Т.Публичный:  /*защ = Защита.Публичный;*/  break;
      default:
        ошибка2(ИДС.ExpectedBaseClasses, сема);
        return основы;
      }
      далее(); // Skip защита attribute.
    LparseBasicType:
      auto начало = сема;
      auto тип = разборБазовогоТипа();
      основы ~= установи(new ТипКлассОснова(защ, тип), начало);
    } while (проверено(Т.Запятая))
    return основы;
  }

  Декларация разборДекларацииИнтерфейса()
  {
    auto начало = сема;
    пропусти(Т.Интерфейс);

    Идентификатор* имя;
    ПараметрыШаблона шпарамы;
    Выражение констрейнт;
    ТипКлассОснова[] основы;
    СложнаяДекларация деклы;

    имя = требуетсяИдентификатор(сооб.ОжидалосьНазваниеИнтерфейса);
```

```
    if (сема.вид == Т.ЛСкобка)
    {
      шпарамы = разборСпискаПараметровШаблона();
      version(D2) констрейнт = разборДополнительногоКонстрейнта();
    }

    if (сема.вид == Т.Двоеточие)
      основы = разборБазовыхКлассов();

    if (основы.length == 0 && проверено(Т.ТочкаЗапятая))
    {}
    else if (сема.вид == Т.ЛФСкобка)
      деклы = разборТелаДефинициИДекларации();
    else
      ошибка2(сооб.ОжидалосьТелоИнтерфейса, сема);

    Декларация d = new ДекларацияИнтерфейса(имя, /+шпарамы, +/основы, деклы);
    if (шпарамы)
      d = поместиДекларациюВнутреннегоШаблона(начало, имя, d, шпарамы,
констрейнт);
    return d;
  }

  Декларация разборДекларацииСтруктурыИлиСоюза()
  {
    assert(сема.вид == Т.Структура || сема.вид == Т.Союз);
    auto начало = сема;
    пропусти(сема.вид);

    Идентификатор* имя;
    ПараметрыШаблона шпарамы;
    Выражение констрейнт;
    СложнаяДекларация деклы;

    имя = дополнительныйИдентификатор();

    if (имя && сема.вид == Т.ЛСкобка)
    {
      шпарамы = разборСпискаПараметровШаблона();
      version(D2) констрейнт = разборДополнительногоКонстрейнта();
    }

    if (имя && проверено(Т.ТочкаЗапятая))
    {}
    else if (сема.вид == Т.ЛФСкобка)
      деклы = разборТелаДефинициИДекларации();
    else
      ошибка2(начало.вид == Т.Структура ?
              сооб.ОжидалосьТелоСтруктуры :
              сооб.ОжидалосьТелоСоюза, сема);

    Декларация d;
    if (начало.вид == Т.Структура)
    {
      auto sd = new ДекларацияСтруктуры(имя, /+шпарамы, +/деклы);
      sd.установиРазмерРаскладки(this.размерРаскладки);
      d = sd;
    }
    else
      d = new ДекларацияСоюза(имя, /+шпарамы, +/деклы);

    if (шпарамы)
```

```
      d = поместиДекларациюВнутреннегоШаблона(начало, имя, d, шпарамы,
констрейнт);
    return d;
  }

  Декларация разборДекларацииКонструктора()
  {
    пропусти(Т.Этот);
    auto параметры = разборСпискаПараметров();
    auto телоФунк = разборТелаФункции();
    return new ДекларацияКонструктора(параметры, телоФунк);
  }

  Декларация разборДекларацииДеструктора()
  {
    пропусти(Т.Тильда);
    требуется(Т.Этот);
    требуется(Т.ЛСкобка);
    требуется(Т.ПСкобка);
    auto телоФунк = разборТелаФункции();
    return new ДекларацияДеструктора(телоФунк);
  }

  Декларация разборДекларацииСтатичКонструктора()
  {
    пропусти(Т.Статический);
    пропусти(Т.Этот);
    требуется(Т.ЛСкобка);
    требуется(Т.ПСкобка);
    auto телоФунк = разборТелаФункции();
    return new ДекларацияСтатическогоКонструктора(телоФунк);
  }

  Декларация разборДекларацииСтатичДеструктора()
  {
    пропусти(Т.Статический);
    пропусти(Т.Тильда);
    требуется(Т.Этот);
    требуется(Т.ЛСкобка);
    требуется(Т.ПСкобка);
    auto телоФунк = разборТелаФункции();
    return new ДекларацияСтатическогоДеструктора(телоФунк);
  }

  Декларация разборДекларацииИнварианта()
  {
    пропусти(Т.Инвариант);
    // Optional () for getting ready porting в D 2.0
    if (проверено(Т.ЛСкобка))
      требуется(Т.ПСкобка);
    auto телоФунк = разборТелаФункции();
    return new ДекларацияИнварианта(телоФунк);
  }

  Декларация разборДекларацииЮниттеста()
  {
    пропусти(Т.Юниттест);
    auto телоФунк = разборТелаФункции();
    return new ДекларацияЮниттеста(телоФунк);
  }

  Сема* разборИдентИлиЦел()
  {
```

```
  if (проверено(Т.Цел32) || проверено(Т.Идентификатор))
    return this.предыдущСема;
  ошибка2(сооб.ОжидалсяИдентИлиЦел, сема);
  return null;
}

Сема* разборУсловияВерсии()
{
version(D2)
{
  if (проверено(Т.Юниттест))
    return this.предыдущСема;
}
  return разборИдентИлиЦел();
}

Декларация разборДекларацииОтладки()
{
  пропусти(Т.Отладка);

  Сема* спец;
  Сема* услов;
  Декларация деклы, деклыИначе;

  if (проверено(Т.Присвоить))
  { // debug = Integer ;
    // debug = Идентификатор ;
    спец = разборИдентИлиЦел();
    требуется(Т.ТочкаЗапятая);
  }
  else
  { // ( Condition )
    if (проверено(Т.ЛСкобка))
    {
      услов = разборИдентИлиЦел();
      требуется(Т.ПСкобка);
    }
    // debug DeclarationsBlock
    // debug ( Condition ) DeclarationsBlock
    деклы = разборБлокаДеклараций();
    // else DeclarationsBlock
    if (проверено(Т.Иначе))
      деклыИначе = разборБлокаДеклараций();
  }

  return new ДекларацияОтладки(спец, услов, деклы, деклыИначе);
}

Декларация парсируйДекларациюВерсии()
{
  пропусти(Т.Версия);

  Сема* спец;
  Сема* услов;
  Декларация деклы, деклыИначе;

  if (проверено(Т.Присвоить))
  { // version = Integer ;
    // version = Идентификатор ;
    спец = разборИдентИлиЦел();
    требуется(Т.ТочкаЗапятая);
  }
  else
```

```
  { // ( Condition )
    требуется(Т.ЛСкобка);
    услов = разборУсловияВерсии();
    требуется(Т.ПСкобка);
    // version ( Condition ) DeclarationsBlock
    деклы = разборБлокаДеклараций();
    // else DeclarationsBlock
    if (проверено(Т.Иначе))
      деклыИначе = разборБлокаДеклараций();
  }

  return new ДекларацияВерсии(спец, услов, деклы, деклыИначе);
}

Декларация парсируйДекларациюСтатичЕсли()
{
  пропусти(Т.Статический);
  пропусти(Т.Если);

  Выражение условие;
  Декларация деклыЕсли, деклыИначе;

  auto leftParen = сема;
  требуется(Т.ЛСкобка);
  условие = разборВыраженияПрисвой();
  требуетсяЗакрыв(Т.ПСкобка, leftParen);

  деклыЕсли = разборБлокаДеклараций();

  if (проверено(Т.Иначе))
    деклыИначе = разборБлокаДеклараций();

  return new ДекларацияСтатическогоЕсли(условие, деклыЕсли, деклыИначе);
}

Декларация парсируйДекларациюСтатичАссерта()
{
  пропусти(Т.Статический);
  пропусти(Т.Подтвердить);
  Выражение условие, сообщение;
  auto leftParen = сема;
  требуется(Т.ЛСкобка);
  условие = разборВыраженияПрисвой();
  if (проверено(Т.Запятая))
    сообщение = разборВыраженияПрисвой();
  требуетсяЗакрыв(Т.ПСкобка, leftParen);
  требуется(Т.ТочкаЗапятая);
  return new ДекларацияСтатическогоПодтверди(условие, сообщение);
}

Декларация парсируйДекларациюШаблона()
{
  пропусти(Т.Шаблон);
  auto имя = требуетсяИдентификатор(сооб.ОжидалосьНазваниеШаблона);
  auto шпарамы = разборСпискаПараметровШаблона();
  auto констрейнт = разборДополнительногоКонстрейнта();
  auto деклы = разборТелаДефинцииДекларации();
  return new ДекларацияШаблона(имя, шпарамы, констрейнт, деклы);
}

Декларация парсируйДекларациюНов()
{
  пропусти(Т.Нов);
```

```
    auto параметры = разборСпискаПараметров();
    auto телоФунк = разборТелаФункции();
    return new ДекларацияНов(параметры, телоФунк);
  }

  Декларация парсируйДекларациюУдалить()
  {
    пропусти(Т.Удалить);
    auto параметры = разборСпискаПараметров();
    auto телоФунк = разборТелаФункции();
    return new ДекларацияУдали(параметры, телоФунк);
  }

  Тип parseTypeofType()
  {
    auto начало = сема;
    пропусти(Т.Типа);
    auto leftParen = сема;
    требуется(Т.ЛСкобка);
    Тип тип;
    switch (сема.вид)
    {
    version(D2)
    {
    case Т.Итог:
      далее();
      тип = new ТТип();
      break;
    }
    default:
      тип = new ТТип(разборВыражения());
    }
    требуетсяЗакрыв(Т.ПСкобка, leftParen);
    установи(тип, начало);
    return тип;
  }

  /// Parses a ДекларацияСмеси or ИнструкцияСмесь.
  /// $(PRE
  /// TemplateMixin :=
  ///         mixin ( ВыражениеПрисвой ) ;
  ///         mixin TemplateIdentifier ;
  ///         mixin TemplateIdentifier MixinIdentifier ;
  ///         mixin TemplateIdentifier !( АргументыШаблона ) ;
  ///         mixin TemplateIdentifier !( АргументыШаблона ) MixinIdentifier
;
  /// )
  Класс парсируйМиксин(Класс)()
  {
  static assert(is(Класс == ДекларацияСмеси) || is(Класс ==
ИнструкцияСмесь));
    пропусти(Т.Смесь);

  static if (is(Класс == ДекларацияСмеси))
  {
    if (проверено(Т.ЛСкобка))
    {
      auto leftParen = сема;
      auto в = разборВыраженияПрисвой();
      требуетсяЗакрыв(Т.ПСкобка, leftParen);
      требуется(Т.ТочкаЗапятая);
      return new ДекларацияСмеси(в);
    }
```

```
  }

    auto начало = сема;
    Выражение в;
    Идентификатор* идентСмеси;

    if (проверено(Т.Точка))
      в = установи(new
ВыражениеМасштабМодуля(разборВыраженияИдентификатора()), начало);
    else
      в = разборВыраженияИдентификатора();

    while (проверено(Т.Точка))
      в = установи(new ВыражениеТочка(в, разборВыраженияИдентификатора()),
начало);

    идентСмеси = дополнительныйИдентификатор();
    требуется(Т.ТочкаЗапятая);

    return new Класс(в, идентСмеси);
  }


/+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  |                          Инструкция parsing methods
|

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+/

  СложнаяИнструкция разборИнструкций()
  {
    auto начало = сема;
    требуется(Т.ЛФСкобка);
    auto инструкции = new СложнаяИнструкция();
    while (сема.вид != Т.ПФСкобка && сема.вид != Т.КФ)
      инструкции ~= разборИнструкции();
    требуетсяЗакрыв(Т.ПФСкобка, начало);
    return установи(инструкции, начало);
  }

  /// Parses a Инструкция.
  Инструкция разборИнструкции()
  {
    auto начало = сема;
    Инструкция s;
    Декларация d;

    if (сема.интегральныйТип_ли)
    {
      d = разборПеременнойИлиФункции();
      goto LreturnDeclarationStatement;
    }

    switch (сема.вид)
    {
    case Т.Расклад:
      бцел размер = разборАтрибутаАлайн();
      // Retктict align attribute в structs in parsing phase.
      ДекларацияСтруктуры structDecl;
      if (сема.вид == Т.Структура)
      {
        auto begin2 = сема;
```

```
        structDecl =
разборДекларацииСтруктурыИлиСоюза().в!(ДекларацияСтруктуры);
        structDecl.установиРазмерРаскладки(размер);
        установи(structDecl, begin2);
      }
      else
        ожидаемое(Т.Структура);

      d = new ДекларацияРазложи(размер, structDecl ?
cast(Декларация)structDecl : new СложнаяДекларация);
      goto LreturnDeclarationStatement;
      /+ Не applicable for инструкции.
         Т.Приватный, Т.Пакет, Т.Защищённый, Т.Публичный, Т.Экспорт,
         Т.Устаревший, Т.Перепись, Т.Абстрактный,+/
    case Т.Экстерн,
         Т.Окончательный,
         Т.Конст,
         Т.Авто:
         //Т.Масштаб
         //Т.Статический
    случай_разборАтрибута:
      s = разборИнструкцииАтрибута();
      return s;
    case Т.Идентификатор:
      if (возьмиСледщ() == Т.Двоеточие)
      {
        auto идент = сема.идент;
        пропусти(Т.Идентификатор); пропусти(Т.Двоеточие);
        s = new ИнструкцияСМеткой(идент,
разборИнструкцииБезМасштабаИлиПустое());
        break;
      }
      goto case Т.Точка;
    case Т.Точка, Т.Типа:
      бул успех;
      d = пробуй_(delegate {
          return разборПеременнойИлиФункции(КлассХранения.Нет,
                                            Защита.Нет,
                                            ТипКомпоновки.Нет, нет, нет);
        }, успех
      );
      if (успех)
        goto LreturnDeclarationStatement; // Декларация
      else
        goto случай_разборИнструкцииВыражения; // Выражение

    case Т.Если:
      s = разборИнструкцииЕсли();
      break;
    case Т.Пока:
      s = разборИнструкцииПока();
      break;
    case Т.Делай:
      s = разборИнструкцииДелайПока();
      break;
    case Т.При:
      s = разборИнструкцииПри();
      break;
    case Т.Длявсех, Т.Длявсех_реверс:
      s = разборИнструкцииДлявсех();
      break;
    case Т.Щит:
      s = разборИнструкцииЩит();
```

```
  break;
case Т.Реле:
  s = разборИнструкцииРеле();
  break;
case Т.Дефолт:
  s = разборИнструкцииДефолт();
  break;
case Т.Далее:
  s = разборИнструкцииДалее();
  break;
case Т.Всё:
  s = разборИнструкцииВсё();
  break;
case Т.Итог:
  s = разборИнструкцииИтог();
  break;
case Т.Переход:
  s = разборИнструкцииПереход();
  break;
case Т.Для:
  s = разборИнструкцииДля();
  break;
case Т.Синхронизованный:
  s = разборИнструкцииСинхронно();
  break;
case Т.Пробуй:
  s = разборИнструкцииПробуй();
  break;
case Т.Брось:
  s = разборИнструкцииБрось();
  break;
case Т.Масштаб:
  if (возьмиСледщ() != Т.ЛСкобка)
    goto случай_разборАтрибута;
  s = parseScopeGuardStatement();
  break;
case Т.Волатайл:
  s = разборИнструкцииВолатайл();
  break;
case Т.Асм:
  s = parseAsmBlockStatement();
  break;
case Т.Прагма:
  s = разборИнструкцииПрагма();
  break;
case Т.Смесь:
  if (возьмиСледщ() == Т.ЛСкобка)
    goto случай_разборИнструкцииВыражения; // Parse as выражение.
  s = парсируйМиксин!(ИнструкцияСмесь)();
  break;
case Т.Статический:
  switch (возьмиСледщ())
  {
  case Т.Если:
    s = разборИнструкцииСтатичЕсли();
    break;
  case Т.Подтвердить:
    s = разборИнструкцииСтатичПровер();
    break;
  default:
    goto случай_разборАтрибута;
  }
  break;
```

```
case Т.Отладка:
  s = разборИнструкцииОтладка();
  break;
case Т.Версия:
  s = разборИнструкцииВерсия();
  break;
// DeclDef
case Т.Алиас, Т.Типдеф:
  d = разборДефинициидекларации();
  goto LreturnDeclarationStatement;
case Т.Перечень:
version(D2)
{
  if (манифестПеречня_ли())
    goto случай_разборАтрибута;
}
  d = разборДеклараciиПеречня();
  goto LreturnDeclarationStatement;
case Т.Класс:
  d = разборДеклараciиКласса();
  goto LreturnDeclarationStatement;
case Т.Интерфейс:
  d = разборДеклараciиИнтерфейса();
  goto LreturnDeclarationStatement;
case Т.Структура, Т.Союз:
  d = разборДеклараciиСтруктурыИлиСоюза();
  // goto LreturnDeclarationStatement;
LreturnDeclarationStatement:
  установи(d, начало);
  s = new ИнструкцияДекларация(d);
  break;
case Т.ЛФСкобка:
  s = разборИнструкцииМасштаб();
  break;
case Т.ТочкаЗапятая:
  далее();
  s = new ПустаяИнструкция();
  break;
// Parse an ИнструкцияВыражение:
// Токены that старт a PrimaryExpression.
// case Т.Идентификатор, Т.Точка, Т.Типа:
case Т.Этот:
case Т.Супер:
case Т.Нуль:
case Т.Истина, Т.Ложь:
// case Т.Доллар:
case Т.Цел32, Т.Цел64, Т.Бцел32, Т.Бцел64:
case Т.Плав32, Т.Плав64, Т.Плав80,
     Т.Мнимое32, Т.Мнимое64, Т.Мнимое80:
case Т.СимЛитерал:
case Т.Ткст:
case Т.ЛКвСкобка:
// case Т.ЛФСкобка:
case Т.Функция, Т.Делегат:
case Т.Подтвердить:
// case Т.Смесь:
case Т.Импорт:
case Т.Идтипа:
case Т.Является:
case Т.ЛСкобка:
case Т.Трэтс: // D2.0
// Токены that can старт a УнарноеВыражение:
case Т.ИБинарное, Т.ПлюсПлюс, Т.МинусМинус, Т.Умножь, Т.Минус,
```

```
        Т.Плюс, Т.Не, Т.Тильда, Т.Нов, Т.Удалить, Т.Каст:
    случай_разборИнструкцииВыражения:
      s = new ИнструкцияВыражение(разборВыражения());
      требуется(Т.ТочкаЗапятая);
      break;
    default:
      if (сема.спецСема_ли)
        goto случай_разборИнструкцииВыражения;

      if (сема.вид != Т.Доллар)
        // Подтвердить that this isn't a valid выражение.
        assert(delegate бул(){
            бул успех;
            auto выражение = пробуй_(&разборВыражения, успех);
            return успех;
          }() == нет, "Валидное выражение не ожидалось."
        );

      // Report ошибка: it's an illegal statement.
      s = new НелегальнаяИнструкция();
      // Skip в следщ valid сема.
      do
        далее();
      while (!сема.началоИнстр_ли &&
              сема.вид != Т.ПФСкобка &&
              сема.вид != Т.КФ)
      auto текст = Сема.textSpan(начало, this.предыдущСема);
      ошибка(начало, сооб.НелегальнаяИнструкция, текст);
    }
    assert(s !is null);
    установи(s, начало);
    return s;
}

/// $(PRE
/// Parses a ИнструкцияМасштаб.
/// ИнструкцияМасштаб :=
///     NoScopeStatement
/// )
Инструкция разборИнструкцииМасштаб()
{
  return new ИнструкцияМасштаб(разборИнструкцииБезМасштаба());
}

/// $(PRE
/// NoScopeStatement :=
///     NonEmptyStatement
///     BlockStatement
/// BlockStatement :=
///     { }
///     { StatementList }
/// )
Инструкция разборИнструкцииБезМасштаба()
{
  auto начало = сема;
  Инструкция s;
  if (проверено(Т.ЛФСкобка))
  {
    auto ss = new СложнаяИнструкция();
    while (сема.вид != Т.ПФСкобка && сема.вид != Т.КФ)
      ss ~= разборИнструкции();
    требуетсяЗакрыв(Т.ПФСкобка, начало);
    s = установи(ss, начало);
```

```
  }
  else if (сема.вид == Т.ТочкаЗапятая)
  {
    ошибка(сема, сооб.Ожидалась‌НеПустая‌Инструкция);
    далее();
    s = установи(new ПустаяИнструкция(), начало);
  }
  else
    s = разборИнструкции();
  return s;
}

/// $(PRE
/// NoScopeOrEmptyStatement :=
///     ;
///     NoScopeStatement
/// )
Инструкция разборИнструкцииБезМасштабаИлиПустое()
{
  if (проверено(Т.ТочкаЗапятая))
    return установи(new ПустаяИнструкция(), this.предыдущСема);
  else
    return разборИнструкцииБезМасштаба();
}

Инструкция разборИнструкцииАтрибута()
{
  КлассХранения кхр, stc_tmp;
  ТипКомпоновки предш_типКомпоновки;

  Декларация разбор() // Nested function.
  {
    auto начало = сема;
    Декларация декл;
    switch (сема.вид)
    {
    case Т.Экстерн:
      if (возьмиСледщ() != Т.ЛСкобка)
      {
        stc_tmp = КлассХранения.Экстерн;
        goto Lcommon;
      }

      далее();
      auto типКомпоновки = разборТипаКомпоновки();
      проверьТипКомпоновки(предш_типКомпоновки, типКомпоновки, начало);

      декл = new ДекларацияКомпоновки(типКомпоновки, разбор());
      break;
    case Т.Статический:
      stc_tmp = КлассХранения.Статический;
      goto Lcommon;
    case Т.Окончательный:
      stc_tmp = КлассХранения.Окончательный;
      goto Lcommon;
    case Т.Конст:
    version(D2)
    {
      if (возьмиСледщ() == Т.ЛСкобка)
        goto случай_Декларация;
    }
      stc_tmp = КлассХранения.Конст;
      goto Lcommon;
```

```
      version(D2)
      {
      case Т.Инвариант: // D 2.0
        if (возьмиСледщ() == Т.ЛСкобка)
          goto случай_Декларация;
        stc_tmp = КлассХранения.Инвариант;
        goto Lcommon;
      case Т.Перечень: // D 2.0
        if (!манифестПеречня_ли())
        { // A normal enum declaration.
          декл = разборДеклорацииПеречня();
          // NB: this must be similar в the код at the конец of
          //     разборДефинициИДекларации().
          декл.установиЗащиту(this.защита);
          декл.установиКлассХранения(кхр);
          установи(декл, начало);
          return декл;
        }
        // enum as КлассХранения.
        stc_tmp = КлассХранения.Манифест;
        goto Lcommon;
      }
      case Т.Авто:
        stc_tmp = КлассХранения.Авто;
        goto Lcommon;
      case Т.Масштаб:
        stc_tmp = КлассХранения.Масштаб;
        goto Lcommon;
      Lcommon:
        // Issue ошибка if redundant.
        if (кхр & stc_tmp)
          ошибка2(ИДС.ПовторяющийсяКлассХранения, сема);
        else
          кхр |= stc_tmp;

        далее();
        декл = new ДекларацияКлассаХранения(stc_tmp, разбор());
        break;
      case Т.Класс, Т.Интерфейс, Т.Структура, Т.Союз, Т.Алиас, Т.Типдеф:
        декл = разборДефинициИДекларации();
        декл.установиЗащиту(Защита.Нет);
        декл.установиКлассХранения(КлассХранения.Нет);
        return декл;
      default:
      случай_Декларация:
        return разборПеременнойИлиФункции(кхр, Защита.Нет,
предщ_типКомпоновки, да);
      }
      return установи(декл, начало);
    }
    return new ИнструкцияДекларация(разбор());
  }

  Инструкция разборИнструкцииЕсли()
  {
    пропусти(Т.Если);

    Инструкция переменная;
    Выражение условие;
    Инструкция телоЕсли, телоИначе;

    auto leftParen = сема;
    требуется(Т.ЛСкобка);
```

```
  Идентификатор* идент;
  auto начало = сема; // При старт of ДекларацияАвто or normal Декларация.
  // auto Идентификатор = Выражение
  if (проверено(Т.Авто))
  {
    идент = требуетсяИдентификатор(сооб.ОжидалосьНазваниеПеременной);
    требуется(Т.Присвоить);
    auto иниц = разборВыражения();
    auto v = new ДекларацияПеременных(null, [идент], [иниц]);
    установи(v, начало.следщНепроб);
    auto d = new ДекларацияКлассаХранения(КлассХранения.Авто, v);
    установи(d, начало);
    переменная = new ИнструкцияДекларация(d);
    установи(переменная, начало);
  }
  else
  { // Declarator = Выражение
    Тип parseDeclaratorAssign()
    {
      auto тип = разборДекларатора(идент);
      требуется(Т.Присвоить);
      return тип;
    }
    бул успех;
    auto тип = пробуй_(&parseDeclaratorAssign, успех);
    if (успех)
    {
      auto иниц = разборВыражения();
      auto v = new ДекларацияПеременных(тип, [идент], [иниц]);
      установи(v, начало);
      переменная = new ИнструкцияДекларация(v);
      установи(переменная, начало);
    }
    else
      условие = разборВыражения();
  }
  требуетсяЗакрыв(Т.ПСкобка, leftParen);
  телоЕсли = разборИнструкцииМасштаб();
  if (проверено(Т.Иначе))
    телоИначе = разборИнструкцииМасштаб();
  return new ИнструкцияЕсли(переменная, условие, телоЕсли, телоИначе);
}

Инструкция разборИнструкцииПока()
{
  пропусти(Т.Пока);
  auto leftParen = сема;
  требуется(Т.ЛСкобка);
  auto условие = разборВыражения();
  требуетсяЗакрыв(Т.ПСкобка, leftParen);
  return new ИнструкцияПока(условие, разборИнструкцииМасштаб());
}

Инструкция разборИнструкцииДелайПока()
{
  пропусти(Т.Делай);
  auto телоДелай = разборИнструкцииМасштаб();
  требуется(Т.Пока);
  auto leftParen = сема;
  требуется(Т.ЛСкобка);
  auto условие = разборВыражения();
  требуетсяЗакрыв(Т.ПСкобка, leftParen);
```

```
    return new ИнструкцияДелайПока(условие, телоДелай);
  }

  Инструкция разборИнструкцииПри()
  {
    пропусти(Т.При);

    Инструкция иниц, телоПри;
    Выражение условие, инкремент;

    auto leftParen = сема;
    требуется(Т.ЛСкобка);
    if (!проверено(Т.ТочкаЗапятая))
      иниц = разборИнструкцииБезМасштаба();
    if (сема.вид != Т.ТочкаЗапятая)
      условие = разборВыражения();
    требуется(Т.ТочкаЗапятая);
    if (сема.вид != Т.ПСкобка)
      инкремент = разборВыражения();
    требуетсяЗакрыв(Т.ПСкобка, leftParen);
    телоПри = разборИнструкцииМасштаб();
    return new ИнструкцияПри(иниц, условие, инкремент, телоПри);
  }

  Инструкция разборИнструкцииДлявсех()
  {
    assert(сема.вид == Т.Длявсех || сема.вид == Т.Длявсех_реверс);
    ТОК лекс = сема.вид;
    далее();

    auto парамы = new Параметры;
    Выражение в; // Агрегат or LwrExpression

    auto leftParen = сема;
    требуется(Т.ЛСкобка);
    auto paramsBegin = сема;
    do
    {
      auto paramBegin = сема;
      КлассХранения кхр;
      Тип тип;
      Идентификатор* идент;

      switch (сема.вид)
      {
      case Т.Реф, Т.Вховых:
        кхр = КлассХранения.Реф;
        далее();
        // fall through
      case Т.Идентификатор:
        auto следщ = возьмиСледщ();
        if (следщ == Т.Запятая || следщ == Т.ТочкаЗапятая || следщ ==
Т.ПСкобка)
        {
          идент = требуетсяИдентификатор(сооб.ОжидалосьНазваниеПеременной);
          break;
        }
        // fall through
      default:
        тип = разборДекларатора(идент);
      }

      парамы ~= установи(new Параметр(кхр, тип, идент, null), paramBegin);
```

```
    } while (проверено(Т.Запятая))
    установи(парамы, paramsBegin);
    требуется(Т.ТочкаЗапятая);
    в = разборВыражения();
  version(D2)
  { //Длявсех (ForeachType; LwrExpression .. UprExpression )
ИнструкцияМасштаб
    if (проверено(Т.Срез))
    {
      // if (парамы.length != 1)
        // ошибка(ИДС.XYZ); // TODO: issue ошибка сооб
      auto верхний = разборВыражения();
      требуетсяЗакрыв(Т.ПСкобка, leftParen);
      auto телоПри = разборИнструкцииМасштаб();
      return new ИнструкцияДиапазонСКаждым(лекс, парамы, в, верхний,
телоПри);
    }
  }
    // Длявсех (ForeachTypeList; Агрегат) ИнструкцияМасштаб
    требуетсяЗакрыв(Т.ПСкобка, leftParen);
    auto телоПри = разборИнструкцииМасштаб();
    return new ИнструкцияСКаждым(лекс, парамы, в, телоПри);
  }

  Инструкция разборИнструкцииЩит()
  {
    пропусти(Т.Щит);
    auto leftParen = сема;
    требуется(Т.ЛСкобка);
    auto условие = разборВыражения();
    требуетсяЗакрыв(Т.ПСкобка, leftParen);
    auto телоЩит = разборИнструкцииМасштаб();
    return new ИнструкцияЩит(условие, телоЩит);
  }

  /// Helper function for parsing the body of a default or case statement.
  Инструкция parseCaseOrDefaultBody()
  {
    // Этот function is similar в разборИнструкцииБезМасштаба()
    auto начало = сема;
    auto s = new СложнаяИнструкция();
    while (сема.вид != Т.Реле &&
           сема.вид != Т.Дефолт &&
           сема.вид != Т.ПФСкобка &&
           сема.вид != Т.КФ)
      s ~= разборИнструкции();
    установи(s, начало);
    return установи(new ИнструкцияМасштаб(s), начало);
  }

  Инструкция разборИнструкцииРеле()
  {
    пропусти(Т.Реле);
    auto значения = разборСпискаВыражений();
    требуется(Т.Двоеточие);
    auto телоРеле = parseCaseOrDefaultBody();
    return new ИнструкцияРеле(значения, телоРеле);
  }

  Инструкция разборИнструкцииДефолт()
  {
    пропусти(Т.Дефолт);
    требуется(Т.Двоеточие);
```

```
  auto телоДефолта = parseCaseOrDefaultBody();
  return new ИнструкцияДефолт(телоДефолта);
}

Инструкция разборИнструкцииДалее()
{
  пропусти(Т.Далее);
  auto идент = дополнительныйИдентификатор();
  требуется(Т.ТочкаЗапятая);
  return new ИнструкцияДалее(идент);
}

Инструкция разборИнструкцииВсё()
{
  пропусти(Т.Всё);
  auto идент = дополнительныйИдентификатор();
  требуется(Т.ТочкаЗапятая);
  return new ИнструкцияВсё(идент);
}

Инструкция разборИнструкцииИтог()
{
  пропусти(Т.Итог);
  Выражение выр;
  if (сема.вид != Т.ТочкаЗапятая)
    выр = разборВыражения();
  требуется(Т.ТочкаЗапятая);
  return new ИнструкцияИтог(выр);
}

Инструкция разборИнструкцииПереход()
{
  пропусти(Т.Переход);
  Идентификатор* идент;
  Выражение выРеле;
  switch (сема.вид)
  {
  case Т.Реле:
    идент = сема.идент;
    далее();
    if (сема.вид == Т.ТочкаЗапятая)
      break;
    выРеле = разборВыражения();
    break;
  case Т.Дефолт:
    идент = сема.идент;
    далее();
    break;
  default:
    идент = требуетсяИдентификатор(сооб.ОжидалсяИдентификатор);
  }
  требуется(Т.ТочкаЗапятая);
  return new ИнструкцияПереход(идент, выРеле);
}

Инструкция разборИнструкцииДля()
{
  пропусти(Т.Для);
  auto leftParen = сема;
  требуется(Т.ЛСкобка);
  auto выр = разборВыражения();
  требуетсяЗакрыв(Т.ПСкобка, leftParen);
  return new ИнструкцияДля(выр, разборИнструкцииМасштаб());
```

```
  }

  Инструкция разборИнструкцииСинхронно()
  {
    пропусти(Т.Синхронизованный);
    Выражение выр;
    if (проверено(Т.ЛСкобка))
    {
      auto leftParen = this.предыдущСема;
      выр = разборВыражения();
      требуетсяЗакрыв(Т.ПСкобка, leftParen);
    }
    return new ИнструкцияСинхр(выр, разборИнструкцииМасштаб());
  }

  Инструкция разборИнструкцииПробуй()
  {
    auto начало = сема;
    пропусти(Т.Пробуй);

    auto телоПробуй = разборИнструкцииМасштаб();
    ИнструкцияЛови[] телаЛови;
    ИнструкцияИтожь finBody;

    while (проверено(Т.Кэтч))
    {
      Параметр парам;
      if (проверено(Т.ЛСкобка))
      {
        auto begin2 = сема;
        Идентификатор* идент;
        auto тип = разборДекларатора(идент, да);
        парам = new Параметр(КлассХранения.Нет, тип, идент, null);
        установи(парам, begin2);
        требуется(Т.ПСкобка);
      }
      телаЛови ~= установи(new ИнструкцияЛови(парам,
разборИнструкцииБезМасштаба()), начало);
      if (парам is null)
        break; // Этот is a LastCatch
      начало = сема;
    }

    if (проверено(Т.Finally))
      finBody = установи(new ИнструкцияИтожь(разборИнструкцииБезМасштаба()),
предыдущСема);

    if (телаЛови.length == 0 && finBody is null)
      assert(начало.вид == Т.Пробуй), ошибка(начало,
сооб.НеДостаетCatchИлиFinally);

    return new ИнструкцияПробуй(телоПробуй, телаЛови, finBody);
  }

  Инструкция разборИнструкцииБрось()
  {
    пропусти(Т.Брось);
    auto выр = разборВыражения();
    требуется(Т.ТочкаЗапятая);
    return new ИнструкцияБрось(выр);
  }

  Инструкция parseScopeGuardStatement()
```

```
  {
    пропусти(Т.Масштаб);
    пропусти(Т.ЛСкобка);
    auto условие =
требуетсяИдентификатор(сооб.ОжидалсяИдентификаторМасштаба);
    if (условие)
      switch (условие.видИд)
      {
      case ВИД.выход, ВИД.успех, ВИД.сбой:
        break;
      default:
        ошибка2(сооб.НеверныйИдентификаторМасштаба, this.предыдущСема);
      }
    требуется(Т.ПСкобка);
    Инструкция телоМасштаба;
    if (сема.вид == Т.ЛФСкобка)
      телоМасштаба = разборИнструкцииМасштаб();
    else
      телоМасштаба = разборИнструкцииБезМасштаба();
    return new ИнструкцияСтражМасштаба(условие, телоМасштаба);
  }

  Инструкция разборИнструкцииВолатайл()
  {
    пропусти(Т.Волатайл);
    Инструкция телоЛетучего;
    if (сема.вид == Т.ТочкаЗапятая)
      далее();
    else if (сема.вид == Т.ЛФСкобка)
      телоЛетучего = разборИнструкцииМасштаб();
    else
      телоЛетучего = разборИнструкции();
    return new ИнструкцияЛетучее(телоЛетучего);
  }

  Инструкция разборИнструкцииПрагма()
  {
    пропусти(Т.Прагма);

    Идентификатор* идент;
    Выражение[] арги;
    Инструкция телоПрагмы;

    auto leftParen = сема;
    требуется(Т.ЛСкобка);
    идент = требуетсяИдентификатор(сооб.ОжидалсяИдентификаторПрагмы);

    if (проверено(Т.Запятая))
      арги = разборСпискаВыражений();
    требуетсяЗакрыв(Т.ПСкобка, leftParen);

    телоПрагмы = разборИнструкцииБезМасштабаИлиПустое();

    return new ИнструкцияПрагма(идент, арги, телоПрагмы);
  }

  Инструкция разборИнструкцииСтатичЕсли()
  {
    пропусти(Т.Статический);
    пропусти(Т.Если);
    Выражение условие;
    Инструкция телоЕсли, телоИначе;
```

```
  auto leftParen = сема;
  требуется(Т.ЛСкобка);
  условие = разборВыражения();
  требуетсяЗакрыв(Т.ПСкобка, leftParen);
  телоЕсли = разборИнструкцииБезМасштаба();
  if (проверено(Т.Иначе))
    телоИначе = разборИнструкцииБезМасштаба();
  return new ИнструкцияСтатическоеЕсли(условие, телоЕсли, телоИначе);
}

Инструкция разборИнструкцииСтатичПровер()
{
  пропусти(Т.Статический);
  пропусти(Т.Подтвердить);
  Выражение условие, сообщение;

  требуется(Т.ЛСкобка);
  условие = разборВыраженияПрисвой(); // Condition.
  if (проверено(Т.Запятая))
    сообщение = разборВыраженияПрисвой(); // Ошибка сообщение.
  требуется(Т.ПСкобка);
  требуется(Т.ТочкаЗапятая);
  return new ИнструкцияСтатическоеПодтверди(условие, сообщение);
}

Инструкция разборИнструкцииОтладка()
{
  пропусти(Т.Отладка);
  Сема* услов;
  Инструкция телоОтладки, телоИначе;

  // ( Condition )
  if (проверено(Т.ЛСкобка))
  {
    услов = разборИдентИлиЦел();
    требуется(Т.ПСкобка);
  }
  // debug Инструкция
  // debug ( Condition ) Инструкция
  телоОтладки = разборИнструкцииБезМасштаба();
  // else Инструкция
  if (проверено(Т.Иначе))
    телоИначе = разборИнструкцииБезМасштаба();

  return new ИнструкцияОтладка(услов, телоОтладки, телоИначе);
}

Инструкция разборИнструкцииВерсия()
{
  пропусти(Т.Версия);
  Сема* услов;
  Инструкция телоВерсии, телоИначе;

  // ( Condition )
  требуется(Т.ЛСкобка);
  услов = разборУсловияВерсии();
  требуется(Т.ПСкобка);
  // version ( Condition ) Инструкция
  телоВерсии = разборИнструкцииБезМасштаба();
  // else Инструкция
  if (проверено(Т.Иначе))
    телоИначе = разборИнструкцииБезМасштаба();
```

```
    return new ИнструкцияВерсия(услов, телоВерсии, телоИначе);
  }


/+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  |                          Assembler parsing methods
|

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+/

  /// Parses an ИнструкцияБлокАсм.
  Инструкция parseAsmBlockStatement()
  {
    пропусти(Т.Асм);
    auto леваяФСкобка = сема;
    требуется(Т.ЛФСкобка);
    auto ss = new СложнаяИнструкция;
    while (сема.вид != Т.ПФСкобка && сема.вид != Т.КФ)
      ss ~= parseAsmStatement();
    требуетсяЗакрыв(Т.ПФСкобка, леваяФСкобка);
    return new ИнструкцияБлокАсм(ss);
  }

  Инструкция parseAsmStatement()
  {
    auto начало = сема;
    Инструкция s;
    Идентификатор* идент;
    switch (сема.вид)
    {
    // Keywords that are valid opcodes.
    case Т.Вхо, Т.Цел, Т.Вых:
      идент = сема.идент;
      далее();
      goto LOpcode;
    case Т.Идентификатор:
      идент = сема.идент;
      далее();
      if (проверено(Т.Двоеточие))
      { // Идентификатор : ИнструкцияАсм
        s = new ИнструкцияСМеткой(идент, parseAsmStatement());
        break;
      }

    LOpcode:
      // Opcode ;
      // Opcode Operands ;
      // Opcode
      //     Идентификатор
      Выражение[] es;
      if (сема.вид != Т.ТочкаЗапятая)
        do
          es ~= parseAsmExpression();
        while (проверено(Т.Запятая))
      требуется(Т.ТочкаЗапятая);
      s = new ИнструкцияАсм(идент, es);
      break;
    case Т.Расклад:
      // align Integer;
      далее();
      цел число = -1;
      if (сема.вид == Т.Цел32)
        (число = сема.цел_), пропусти(Т.Цел32);
```

```
    else
      ошибка2(сооб.ExpectedIntegerAfterAlign, сема);
    требуется(Т.ТочкаЗапятая);
    s = new ИнструкцияАсмРасклад(число);
    break;
  case Т.ТочкаЗапятая:
    s = new ПустаяИнструкция();
    далее();
    break;
  default:
    s = new ИнструкцияНелегальныйАсм();
    // Skip в следщ valid сема.
    do
      далее();
    while (!сема.началоАсмИнстр_ли &&
            сема.вид != Т.ПФСкобка &&
            сема.вид != Т.КФ)
    auto текст = Сема.textSpan(начало, this.предыдущСема);
    ошибка(начало, сооб.ИнструкцияНелегальныйАсм, текст);
  }
  установи(s, начало);
  return s;
}

Выражение parseAsmExpression()
{
  auto начало = сема;
  auto в = parseAsmOrOrExpression();
  if (проверено(Т.Вопрос))
  {
    auto лекс = this.предыдущСема;
    auto iftrue = parseAsmExpression();
    требуется(Т.Двоеточие);
    auto iffalse = parseAsmExpression();
    в = new ВыражениеУсловия(в, iftrue, iffalse, лекс);
    установи(в, начало);
  }
  // TODO: create AsmExpression that contains в?
  return в;
}

Выражение parseAsmOrOrExpression()
{
  alias parseAsmAndAndExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();
  while (сема.вид == Т.ИлиЛогическое)
  {
    auto лекс = сема;
    далее();
    в = new ВыражениеИлиИли(в, разборСледующего(), лекс);
    установи(в, начало);
  }
  return в;
}

Выражение parseAsmAndAndExpression()
{
  alias parseAsmOrExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();
  while (сема.вид == Т.ИЛогическое)
  {
```

```
    auto лекс = сема;
    далее();
    в = new ВыражениеИИ(в, разборСледующего(), лекс);
    установи(в, начало);
  }
  return в;
}

Выражение parseAsmOrExpression()
{
  alias parseAsmXorExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();
  while (сема.вид == Т.ИлиБинарное)
  {
    auto лекс = сема;
    далее();
    в = new ВыражениеИли(в, разборСледующего(), лекс);
    установи(в, начало);
  }
  return в;
}

Выражение parseAsmXorExpression()
{
  alias parseAsmAndExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();
  while (сема.вид == Т.ИИли)
  {
    auto лекс = сема;
    далее();
    в = new ВыражениеИИли(в, разборСледующего(), лекс);
    установи(в, начало);
  }
  return в;
}

Выражение parseAsmAndExpression()
{
  alias parseAsmCmpExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();
  while (сема.вид == Т.ИБинарное)
  {
    auto лекс = сема;
    далее();
    в = new ВыражениеИ(в, разборСледующего(), лекс);
    установи(в, начало);
  }
  return в;
}

Выражение parseAsmCmpExpression()
{
  alias parseAsmShiftExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();

  auto operator = сема;
  switch (operator.вид)
  {
  case Т.Равно, Т.НеРавно:
```

```
      далее();
      в = new ВыражениеРавно(в, разборСледующего(), operator);
      break;
    case Т.МеньшеРавно, Т.Меньше, Т.БольшеРавно, Т.Больше:
      далее();
      в = new ВыражениеОтнош(в, разборСледующего(), operator);
      break;
    default:
      return в;
    }
    установи(в, начало);
    return в;
  }

  Выражение parseAsmShiftExpression()
  {
    alias parseAsmAddExpression разборСледующего;
    auto начало = сема;
    auto в = разборСледующего();
    while (1)
    {
      auto operator = сема;
      switch (operator.вид)
      {
      case Т.ЛСдвиг:  далее(); в = new ВыражениеЛСдвиг(в, разборСледующего(),
operator); break;
      case Т.ПСдвиг:  далее(); в = new ВыражениеПСдвиг(в, разборСледующего(),
operator); break;
      case Т.URShift: далее(); в = new ВыражениеБПСдвиг(в,
разборСледующего(), operator); break;
      default:
        return в;
      }
      установи(в, начало);
    }
    assert(0);
  }

  Выражение parseAsmAddExpression()
  {
    alias parseAsmMulExpression разборСледующего;
    auto начало = сема;
    auto в = разборСледующего();
    while (1)
    {
      auto operator = сема;
      switch (operator.вид)
      {
      case Т.Плюс:  далее(); в = new ВыражениеПлюс(в, разборСледующего(),
operator); break;
      case Т.Минус: далее(); в = new ВыражениеМинус(в, разборСледующего(),
operator); break;
      // Не allowed in asm
      //case Т.Тильда: далее(); в = new ВыражениеСоедини(в,
разборСледующего(), operator); break;
      default:
        return в;
      }
      установи(в, начало);
    }
    assert(0);
  }
```

```
  Выражение parseAsmMulExpression()
  {
    alias parseAsmPostExpression разборСледующего;
    auto начало = сема;
    auto в = разборСледующего();
    while (1)
    {
      auto operator = сема;
      switch (operator.вид)
      {
      case Т.Умножь: далее(); в = new ВыражениеУмножь(в, разборСледующего(),
operator); break;
      case Т.Деление: далее(); в = new ВыражениеДели(в, разборСледующего(),
operator); break;
      case Т.Модуль: далее(); в = new ВыражениеМод(в, разборСледующего(),
operator); break;
      default:
        return в;
      }
      установи(в, начало);
    }
    assert(0);
  }

  Выражение parseAsmPostExpression()
  {
    auto начало = сема;
    auto в = parseAsmUnaryExpression();
    while (проверено(Т.ЛКвСкобка))
    {
      auto leftBracket = this.предыдущСема;
      в = new ВыражениеАсмПослеСкобки(в, parseAsmExpression());
      требуетсяЗакрыв(Т.ПКвСкобка, leftBracket);
      установи(в, начало);
    }
    return в;
  }

  Выражение parseAsmUnaryExpression()
  {
    auto начало = сема;
    Выражение в;
    switch (сема.вид)
    {
    case Т.Байт,  Т.Крат,  Т.Цел,
         Т.Плав, Т.Дво, Т.Реал:
      goto LAsmTypePrefix;
    case Т.Идентификатор:
      switch (сема.идент.видИд)
      {
      case ВИД.near, ВИД.far,/* "байт",  "крат",  "цел",*/
           ВИД.word, ВИД.dword, ВИД.qword/*, "плав", "дво", "реал"*/:
      LAsmTypePrefix:
        далее();
        if (сема.вид == Т.Идентификатор && сема.идент is Идент.ptr)
          пропусти(Т.Идентификатор);
        else
          ошибка2(ИДС.НайденоИноеЧемОжидалось, "ptr", сема);
        в = new ВыражениеТипАсм(parseAsmExpression());
        break;
      case ВИД.offset:
        далее();
        в = new ВыражениеСмещениеАсм(parseAsmExpression());
```

```
      break;
    case ВИД.seg:
      далее();
      в = new ВыражениеСегАсм(parseAsmExpression());
      break;
    default:
      goto LparseAsmPrimaryExpression;
    }
    break;
  case Т.Минус:
  case Т.Плюс:
    далее();
    в = new ВыражениеЗнак(parseAsmUnaryExpression());
    break;
  case Т.Не:
    далее();
    в = new ВыражениеНе(parseAsmUnaryExpression());
    break;
  case Т.Тильда:
    далее();
    в = new ВыражениеКомп(parseAsmUnaryExpression());
    break;
  case Т.Точка:
    далее();
    в = new ВыражениеМасштабМодуля(разборВыраженияИдентификатора());
    while (проверено(ТОК.Точка))
    {
      в = new ВыражениеТочка(в, разборВыраженияИдентификатора());
      установи(в, начало);
    }
    break;
  default:
  LparseAsmPrimaryExpression:
    в = parseAsmPrimaryExpression();
    return в;
  }
  установи(в, начало);
  return в;
}

Выражение parseAsmPrimaryExpression()
{
  auto начало = сема;
  Выражение в;
  switch (сема.вид)
  {
  case Т.Цел32, Т.Цел64, Т.Бцел32, Т.Бцел64:
    в = new ЦелВыражение(сема);
    далее();
    break;
  case Т.Плав32, Т.Плав64, Т.Плав80,
       Т.Мнимое32, Т.Мнимое64, Т.Мнимое80:
    в = new ВыражениеРеал(сема);
    далее();
    break;
  case Т.Доллар:
    в = new ВыражениеДоллар();
    далее();
    break;
  case Т.ЛКвСкобка:
    // [ AsmExpression ]
    auto leftBracket = сема;
    далее();
```

```
    в = parseAsmExpression();
    требуетсяЗакрыв(Т.ПКвСкобка, leftBracket);
    в = new ВыражениеАсмСкобка(в);
    break;
  case Т.Идентификатор:
    auto регистр = сема.идент;
    switch (регистр.видИд)
    {
    // __LOCAL_SIZE
    case ВИД.__LOCAL_SIZE:
      далее();
      в = new ВыражениеЛокальногоРазмераАсм();
      break;
    // Register
    case ВИД.ST:
      далее();
      // (1) - (7)
      цел число = -1;
      if (проверено(Т.ЛСкобка))
      {
        if (сема.вид == Т.Цел32)
          (число = сема.цел_), пропусти(Т.Цел32);
        else
          ожидаемое(Т.Цел32);
        требуется(Т.ПСкобка);
      }
      в = new ВыражениеАсмРегистр(регистр, число);
      break;
    case ВИД.FS:
      далее();
      // TODO: is the colon-число part optional?
      цел число = -1;
      if (проверено(Т.Двоеточие))
      {
        // :0, :4, :8
        if (сема.вид == Т.Цел32)
          (число = сема.цел_), пропусти(Т.Цел32);
        if (число != 0 && число != 4 && число != 8)
          ошибка2(ИДС.НайденоИноеЧемОжидалось, "0, 4 or 8", сема);
      }
      в = new ВыражениеАсмРегистр(регистр, число);
      break;
    case ВИД.AL, ВИД.AH, ВИД.AX, ВИД.EAX,
         ВИД.BL, ВИД.BH, ВИД.BX, ВИД.EBX,
         ВИД.CL, ВИД.CH, ВИД.CX, ВИД.ECX,
         ВИД.DL, ВИД.DH, ВИД.DX, ВИД.EDX,
         ВИД.BP, ВИД.EBP, ВИД.SP, ВИД.ESP,
         ВИД.DI, ВИД.EDI, ВИД.SI, ВИД.ESI,
         ВИД.ES, ВИД.CS, ВИД.SS, ВИД.DS, ВИД.GS,
         ВИД.CR0, ВИД.CR2, ВИД.CR3, ВИД.CR4,
         ВИД.DR0, ВИД.DR1, ВИД.DR2, ВИД.DR3, ВИД.DR6, ВИД.DR7,
         ВИД.TR3, ВИД.TR4, ВИД.TR5, ВИД.TR6, ВИД.TR7,
         ВИД.MM0, ВИД.MM1, ВИД.MM2, ВИД.MM3,
         ВИД.MM4, ВИД.MM5, ВИД.MM6, ВИД.MM7,
         ВИД.XMM0, ВИД.XMM1, ВИД.XMM2, ВИД.XMM3,
         ВИД.XMM4, ВИД.XMM5, ВИД.XMM6, ВИД.XMM7:
      далее();
      в = new ВыражениеАсмРегистр(регистр);
      break;
    default:
      в = разборВыраженияИдентификатора();
      while (проверено(ТОК.Точка))
      {
```

```
          в = new ВыражениеТочка(в, разборВыраженияИдентификатора());
          установи(в, начало);
        }
      } // конец of switch
      break;
    default:
      ошибка2(ИДС.НайденоИноеЧемОжидалось, "Выражение", сема);
      в = new НелегальноеВыражение();
      if (!пробуем)
      { // Insert a dummy сема and don't consume current one.
        начало = лексер.вставьПустуюСемуПеред(сема);
        this.предыдущСема = начало;
      }
    }
    установи(в, начало);
    return в;
  }


/+~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  |                          Выражение parsing methods
|

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+/

  /// Parses an Выражение.
  Выражение разборВыражения()
  {
    alias разборВыраженияПрисвой разборСледующего;
    auto начало = сема;
    auto в = разборСледующего();
    while (сема.вид == Т.Запятая)
    {
      auto comma = сема;
      далее();
      в = new ВыражениеЗапятая(в, разборСледующего(), comma);
      установи(в, начало);
    }
    return в;
  }

  Выражение разборВыраженияПрисвой()
  {
    alias разборВыраженияПрисвой разборСледующего;
    auto начало = сема;
    auto в = разборУсловнВыражения();
    switch (сема.вид)
    {
    case Т.Присвоить:
      далее(); в = new ВыражениеПрисвой(в, разборСледующего()); break;
    case Т.ЛСдвигПрисвой:
      далее(); в = new ВыражениеПрисвойЛСдвиг(в, разборСледующего()); break;
    case Т.ПСдвигПрисвой:
      далее(); в = new ВыражениеПрисвойПСдвиг(в, разборСледующего()); break;
    case Т.URShiftAssign:
      далее(); в = new ВыражениеПрисвойБПСдвиг(в, разборСледующего()); break;
    case Т.ИлиПрисвой:
      далее(); в = new ВыражениеПрисвойИли(в, разборСледующего()); break;
    case Т.ИПрисвой:
      далее(); в = new ВыражениеПрисвойИ(в, разборСледующего()); break;
    case Т.ПлюсПрисвой:
      далее(); в = new ВыражениеПрисвойПлюс(в, разборСледующего()); break;
    case Т.МинусПрисвой:
```

```
    далее(); в = new ВыражениеПрисвойМинус(в, разборСледующего()); break;
  case Т.ДелениеПрисвой:
    далее(); в = new ВыражениеПрисвойДел(в, разборСледующего()); break;
  case Т.УмножьПрисвой:
    далее(); в = new ВыражениеПрисвойУмн(в, разборСледующего()); break;
  case Т.МодульПрисвой:
    далее(); в = new ВыражениеПрисвойМод(в, разборСледующего()); break;
  case Т.ИИлиПрисвой:
    далее(); в = new ВыражениеПрисвойИИли(в, разборСледующего()); break;
  case Т.CatAssign:
    далее(); в = new ВыражениеПрисвойСоед(в, разборСледующего()); break;
  default:
    return в;
  }
  установи(в, начало);
  return в;
}

Выражение разборУсловнВыражения()
{
  auto начало = сема;
  auto в = parseOrOrExpression();
  if (сема.вид == Т.Вопрос)
  {
    auto лекс = сема;
    далее();
    auto iftrue = разборВыражения();
    требуется(Т.Двоеточие);
    auto iffalse = разборУсловнВыражения();
    в = new ВыражениеУсловия(в, iftrue, iffalse, лекс);
    установи(в, начало);
  }
  return в;
}

Выражение parseOrOrExpression()
{
  alias parseAndAndExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();
  while (сема.вид == Т.ИлиЛогическое)
  {
    auto лекс = сема;
    далее();
    в = new ВыражениеИлиИли(в, разборСледующего(), лекс);
    установи(в, начало);
  }
  return в;
}

Выражение parseAndAndExpression()
{
  alias parseOrExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();
  while (сема.вид == Т.ИЛогическое)
  {
    auto лекс = сема;
    далее();
    в = new ВыражениеИИ(в, разборСледующего(), лекс);
    установи(в, начало);
  }
  return в;
```

```
}

Выражение parseOrExpression()
{
  alias parseXorExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();
  while (сема.вид == Т.ИлиБинарное)
  {
    auto лекс = сема;
    далее();
    в = new ВыражениеИли(в, разборСледующего(), лекс);
    установи(в, начало);
  }
  return в;
}

Выражение parseXorExpression()
{
  alias parseAndExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();
  while (сема.вид == Т.ИИли)
  {
    auto лекс = сема;
    далее();
    в = new ВыражениеИИли(в, разборСледующего(), лекс);
    установи(в, начало);
  }
  return в;
}

Выражение parseAndExpression()
{
  alias parseCmpExpression разборСледующего;
  auto начало = сема;
  auto в = разборСледующего();
  while (сема.вид == Т.ИБинарное)
  {
    auto лекс = сема;
    далее();
    в = new ВыражениеИ(в, разборСледующего(), лекс);
    установи(в, начало);
  }
  return в;
}

Выражение parseCmpExpression()
{
  alias parseShiftExpression разборСледующего;
  auto начало = сема;
  auto в = parseShiftExpression();

  auto operator = сема;
  switch (operator.вид)
  {
  case Т.Равно, Т.НеРавно:
    далее();
    в = new ВыражениеРавно(в, разборСледующего(), operator);
    break;
  case Т.Не:
    if (возьмиСледщ() != Т.Является)
      break;
```

```
      далее();
      // fall through
    case Т.Является:
      далее();
      в = new ВыражениеРавенство(в, разборСледующего(), operator);
      break;
    case Т.МеньшеРавно, Т.Меньше, Т.БольшеРавно, Т.Больше,
         Т.Unordered, Т.UorE, Т.UorG, Т.UorGorE,
         Т.UorL, Т.UorLorE, Т.LorEorG, Т.LorG:
      далее();
      в = new ВыражениеОтнош(в, разборСледующего(), operator);
      break;
    case Т.Вхо:
      далее();
      в = new ВыражениеВхо(в, разборСледующего(), operator);
      break;
    default:
      return в;
    }
    установи(в, начало);
    return в;
  }

  Выражение parseShiftExpression()
  {
    alias parseAddExpression разборСледующего;
    auto начало = сема;
    auto в = разборСледующего();
    while (1)
    {
      auto operator = сема;
      switch (operator.вид)
      {
      case Т.ЛСдвиг:  далее(); в = new ВыражениеЛСдвиг(в, разборСледующего(),
operator); break;
      case Т.ПСдвиг:  далее(); в = new ВыражениеПСдвиг(в, разборСледующего(),
operator); break;
      case Т.URShift: далее(); в = new ВыражениеБПСдвиг(в,
разборСледующего(), operator); break;
      default:
        return в;
      }
      установи(в, начало);
    }
    assert(0);
  }

  Выражение parseAddExpression()
  {
    alias parseMulExpression разборСледующего;
    auto начало = сема;
    auto в = разборСледующего();
    while (1)
    {
      auto operator = сема;
      switch (operator.вид)
      {
      case Т.Плюс:  далее(); в = new ВыражениеПлюс(в, разборСледующего(),
operator); break;
      case Т.Минус: далее(); в = new ВыражениеМинус(в, разборСледующего(),
operator); break;
      case Т.Тильда: далее(); в = new ВыражениеСоедини(в, разборСледующего(),
operator); break;
```

```
      default:
        return в;
      }
      установи(в, начало);
    }
    assert(0);
  }

  Выражение parseMulExpression()
  {
    alias parsePostExpression разборСледующего;
    auto начало = сема;
    auto в = разборСледующего();
    while (1)
    {
      auto operator = сема;
      switch (operator.вид)
      {
      case Т.Умножь: далее(); в = new ВыражениеУмножь(в, разборСледующего(),
operator); break;
      case Т.Деление: далее(); в = new ВыражениеДели(в, разборСледующего(),
operator); break;
      case Т.Модуль: далее(); в = new ВыражениеМод(в, разборСледующего(),
operator); break;
      default:
        return в;
      }
      установи(в, начало);
    }
    assert(0);
  }

  Выражение parsePostExpression()
  {
    auto начало = сема;
    auto в = parseUnaryExpression();
    while (1)
    {
      while (проверено(Т.Точка))
      {
        в = new ВыражениеТочка(в, parseNewOrIdentifierExpression());
        установи(в, начало);
      }

      switch (сема.вид)
      {
      case Т.ПлюсПлюс:
        в = new ВыражениеПостИнкр(в);
        break;
      case Т.МинусМинус:
        в = new ВыражениеПостДекр(в);
        break;
      case Т.ЛСкобка:
        в = new ВыражениеВызов(в, разборАргументов());
        goto Lset;
      case Т.ЛКвСкобка:
        // разбор Срез- and ВыражениеИндекс
        auto leftBracket = сема;
        далее();
        // [] is a ВыражениеСрез
        if (сема.вид == Т.ПКвСкобка)
        {
          в = new ВыражениеСрез(в, null, null);
```

```
        break;
      }

      Выражение[] es = [разборВыраженияПрисвой()];

      // [ ВыражениеПрисвой .. ВыражениеПрисвой ]
      if (проверено(Т.Срез))
      {
        в = new ВыражениеСрез(в, es[0], разборВыраженияПрисвой());
        требуетсяЗакрыв(Т.ПКвСкобка, leftBracket);
        goto Lset;
      }

      // [ ExpressionList ]
      if (проверено(Т.Запятая))
         es ~= разборСпискаВыражений();
      требуетсяЗакрыв(Т.ПКвСкобка, leftBracket);

      в = new ВыражениеИндекс(в, es);
      goto Lset;
    default:
      return в;
    }
    далее();
  Lset: // Jumped here в пропусти далее().
    установи(в, начало);
  }
  assert(0);
}

Выражение parseUnaryExpression()
{
  auto начало = сема;
  Выражение в;
  switch (сема.вид)
  {
  case Т.ИБинарное:
    далее();
    в = new ВыражениеАдрес(parseUnaryExpression());
    break;
  case Т.ПлюсПлюс:
    далее();
    в = new ВыражениеПреИнкр(parseUnaryExpression());
    break;
  case Т.МинусМинус:
    далее();
    в = new ВыражениеПреДекр(parseUnaryExpression());
    break;
  case Т.Умножь:
    далее();
    в = new ВыражениеДереф(parseUnaryExpression());
    break;
  case Т.Минус:
  case Т.Плюс:
    далее();
    в = new ВыражениеЗнак(parseUnaryExpression());
    break;
  case Т.Не:
    далее();
    в = new ВыражениеНе(parseUnaryExpression());
    break;
  case Т.Тильда:
    далее();
```

```
    в = new ВыражениеКомп(parseUnaryExpression());
    break;
  case Т.Нов:
    в = разборВыраженияНов();
    return в;
  case Т.Удалить:
    далее();
    в = new ВыражениеУдали(parseUnaryExpression());
    break;
  case Т.Каст:
    требуетсяСледующий(Т.ЛСкобка);
    Тип тип;
    switch (сема.вид)
    {
    version(D2)
    {
    auto begin2 = сема;
    case Т.Конст:
      тип = new ТКонст(null);
      goto case_break;
    case Т.Инвариант:
      тип = new ТИнвариант(null);
    case_break:
      далее();
      установи(тип, begin2);
      break;
    }
    default:
     тип = разборТипа();
    }
    требуется(Т.ПСкобка);
    в = new ВыражениеКаст(parseUnaryExpression(), тип);
    break;
  case Т.ЛСкобка:
    // ( Тип ) . Идентификатор
    Тип parseType_()
    {
      пропусти(Т.ЛСкобка);
      auto тип = разборТипа();
      требуется(Т.ПСкобка);
      требуется(Т.Точка);
      return тип;
    }
    бул успех;
    auto тип = пробуй_(&parseType_, успех);
    if (успех)
    {
      auto идент = требуетсяИдентификатор(сооб.ExpectedIdAfterTypeDot);
      в = new ВыражениеИдТипаТочка(тип, идент);
      break;
    }
    goto default;
  case Т.Точка:
    далее();
    в = new ВыражениеМасштабМодуля(разборВыраженияИдентификатора());
    break;
  default:
    в = parsePrimaryExpression();
    return в;
  }
  assert(в !is null);
  установи(в, начало);
  return в;
```

```
  }

  /// $(PRE
  /// ВыражениеИдентификатор :=
  ///         Идентификатор
  ///         TemplateInstance
  /// TemplateInstance :=
  ///         Идентификатор !( АргументыШаблона )
  /// )
  Выражение разборВыраженияИдентификатора()
  {
    auto начало = сема;
    auto идент = требуетсяИдентификатор(сооб.ОжидалсяИдентификатор);
    Выражение в;
    // Peek for '(' в avoid matching: ид !is ид
    if (сема.вид == Т.Не && возьмиСледщ() == Т.ЛСкобка)
    { // Идентификатор !( АргументыШаблона )
      пропусти(Т.Не);
      auto шпарамы = parseTemplateArguments();
      в = new ВыражениеЭкземплярШаблона(идент, шпарамы);
    }
    else // Идентификатор
      в = new ВыражениеИдентификатор(идент);
    return установи(в, начало);
  }

  Выражение parseNewOrIdentifierExpression()
  {
    return сема.вид == Т.Нов ? разборВыраженияНов() :
разборВыраженияИдентификатора();
  }

  Выражение parsePrimaryExpression()
  {
    auto начало = сема;
    Выражение в;
    switch (сема.вид)
    {
    case Т.Идентификатор:
      в = разборВыраженияИдентификатора();
      return в;
    case Т.Типа:
      в = new ВыражениеТипа(parseTypeofType());
      break;
    case Т.Этот:
      далее();
      в = new ВыражениеЭтот();
      break;
    case Т.Супер:
      далее();
      в = new ВыражениеСупер();
      break;
    case Т.Нуль:
      далее();
      в = new ВыражениеНуль();
      break;
    case Т.Истина, Т.Ложь:
      далее();
      в = new БулевоВыражение(сема.вид == Т.Истина);
      break;
    case Т.Доллар:
      далее();
      в = new ВыражениеДоллар();
```

```
  break;
case Т.Цел32, Т.Цел64, Т.Бцел32, Т.Бцел64:
  в = new ЦелВыражение(сема);
  далее();
  break;
case Т.Плав32, Т.Плав64, Т.Плав80,
     Т.Мнимое32, Т.Мнимое64, Т.Мнимое80:
  в = new ВыражениеРеал(сема);
  далее();
  break;
case Т.СимЛитерал:
  в = new ВыражениеСим(сема.дим_);
  далее();
  break;
case Т.Ткст:
  ткст ткт = сема.ткт;
  сим postfix = сема.pf;
  далее();
  while (сема.вид == Т.Ткст)
  {
    /+if (postfix == 0)
        postfix = сема.pf;
    else+/
    if (сема.pf && сема.pf != postfix)
      ошибка(сема, сооб.StringPostfixMismatch);
    ткт.length = ткт.length - 1; // Exclude '\0'.
    ткт ~= сема.ткт;
    далее();
  }
  switch (postfix)
  {
  case 'w':
    if (естьНеверныйЮ8(ткт, начало))
      goto default;
    в = new ТекстовоеВыражение(drc.Unicode.вЮ16(ткт)); break;
  case 'd':
    if (естьНеверныйЮ8(ткт, начало))
      goto default;
    в = new ТекстовоеВыражение(drc.Unicode.вЮ32(ткт)); break;
  case 'c':
  default:
    // No checking done в allow for binary данные.
    в = new ТекстовоеВыражение(ткт); break;
  }
  break;
case Т.ЛКвСкобка:
  Выражение[] значения;

  далее();
  if (!проверено(Т.ПКвСкобка))
  {
    в = разборВыраженияПрисвой();
    if (проверено(Т.Двоеточие))
      goto LparseAssocArray;
    if (проверено(Т.Запятая))
      значения = [в] ~ разборСпискаВыражений();
    требуетсяЗакрыв(Т.ПКвСкобка, начало);
  }

  в = new ВыражениеЛитералМассива(значения);
  break;

LparseAssocArray:
```

```
      Выражение[] ключи = [в];

      goto LenterLoop;
      do
      {
        ключи ~= разборВыраженияПрисвой();
        требуется(Т.Двоеточие);
      LenterLoop:
        значения ~= разборВыраженияПрисвой();
      } while (проверено(Т.Запятая))
      требуетсяЗакрыв(Т.ПКвСкобка, начало);
      в = new ВыражениеЛитералАМассива(ключи, значения);
      break;
    case Т.ЛФСкобка:
      // DelegateLiteral := { Statements }
      auto телоФунк = разборТелаФункции();
      в = new ВыражениеЛитералФункции(телоФунк);
      break;
    case Т.Функция, Т.Делегат:
      // FunctionLiteral := ("function"|"delegate") Тип? "(" ArgumentList ")"
FunctionBody
      далее(); // Skip function or delegate keyword.
      Тип типВозврата;
      Параметры параметры;
      if (сема.вид != Т.ЛФСкобка)
      {
        if (сема.вид != Т.ЛСкобка) // Optional return тип
          типВозврата = разборТипа();
        параметры = разборСпискаПараметров();
      }
      auto телоФунк = разборТелаФункции();
      в = new ВыражениеЛитералФункции(типВозврата, параметры, телоФунк);
      break;
    case Т.Подтвердить:
      Выражение сооб;
      требуетсяСледующий(Т.ЛСкобка);
      в = разборВыраженияПрисвой();
      if (проверено(Т.Запятая))
        сооб = разборВыраженияПрисвой();
      требуется(Т.ПСкобка);
      в = new ВыражениеПодтверди(в, сооб);
      break;
    case Т.Смесь:
      требуетсяСледующий(Т.ЛСкобка);
      в = разборВыраженияПрисвой();
      требуется(Т.ПСкобка);
      в = new ВыражениеСмесь(в);
      break;
    case Т.Импорт:
      требуетсяСледующий(Т.ЛСкобка);
      в = разборВыраженияПрисвой();
      требуется(Т.ПСкобка);
      в = new ВыражениеИмпорта(в);
      break;
    case Т.Идтипа:
      требуетсяСледующий(Т.ЛСкобка);
      auto тип = разборТипа();
      требуется(Т.ПСкобка);
      в = new ВыражениеИдТипа(тип);
      break;
    case Т.Является:
      далее();
      auto leftParen = сема;
```

```
      требуется(Т.ЛСкобка);

      Тип тип, типСпец;
      Идентификатор* идент; // optional Идентификатор
      Сема* опцСема, спецСема;

      тип = разборДекларатора(идент, да);

      switch (сема.вид)
      {
      case Т.Двоеточие, Т.Равно:
        опцСема = сема;
        далее();
        switch (сема.вид)
        {
        case Т.Типдеф,
             Т.Структура,
             Т.Союз,
             Т.Класс,
             Т.Интерфейс,
             Т.Перечень,
             Т.Функция,
             Т.Делегат,
             Т.Супер,
             Т.Итог:
        case_Const_Invariant:
          спецСема = сема;
          далее();
          break;
        case Т.Конст, Т.Инвариант:
          if (возьмиСледщ() != Т.ЛСкобка)
            goto case_Const_Invariant;
          // Fall through. It's a тип.
        default:
          типСпец = разборТипа();
        }
      default:
      }

      ПараметрыШаблона шпарамы;
    version(D2)
    {
      // is ( Тип Идентификатор : TypeSpecialization , TemplateParameterList
)
      // is ( Тип Идентификатор == TypeSpecialization , TemplateParameterList
)
      if (идент && типСпец && сема.вид == Т.Запятая)
        шпарамы = разборСпискаПараметровШаблона2();
    }
      требуетсяЗакрыв(Т.ПСкобка, leftParen);
      в = new ВыражениеЯвляется(тип, идент, опцСема, спецСема, типСпец,
шпарамы);
      break;
    case Т.ЛСкобка:
      if (семаПослеСкобкиЯвляется(Т.ЛФСкобка)) // Check for "(...) {"
      { // ( ParameterList ) FunctionBody
        auto параметры = разборСпискаПараметров();
        auto телоФунк = разборТелаФункции();
        в = new ВыражениеЛитералФункции(null, параметры, телоФунк);
      }
      else
      { // ( Выражение )
        auto leftParen = сема;
```

```
        пропусти(Т.ЛСкобка);
        в = разборВыражения();
        требуетсяЗакрыв(Т.ПСкобка, leftParen);
        в = new ВыражениеРодит(в);
      }
      break;
    version(D2)
    {
    case Т.Трэтс:
      требуетсяСледующий(Т.ЛСкобка);
      auto ид = требуетсяИдентификатор(сооб.ОжидалсяИдентификатор);
      АргументыШаблона арги;
      if (сема.вид == Т.Запятая)
        арги = parseTemplateArguments2();
      else
        требуется(Т.ПСкобка);
      в = new ВыражениеТрактовки(ид, арги);
      break;
    }
    default:
      if (сема.интегральныйТип_ли)
      { // ИнтегральныйТип . Идентификатор
        auto тип = new ИнтегральныйТип(сема.вид);
        далее();
        установи(тип, начало);
        требуется(Т.Точка);
        auto идент = требуетсяИдентификатор(сооб.ExpectedIdAfterTypeDot);
        в = new ВыражениеИдТипаТочка(тип, идент);
      }
      else if (сема.спецСема_ли)
      {
        в = new ВыражениеСпецСема(сема);
        далее();
      }
      else
      {
        ошибка2(ИДС.НайденоИноеЧемОжидалось, "Выражение", сема);
        в = new НелегальноеВыражение();
        if (!пробуем)
        { // Insert a dummy сема and don't consume current one.
          начало = лексер.вставьПустуюСемуПеред(сема);
          this.предыдущСема = начало;
        }
      }
    }
    установи(в, начало);
    return в;
  }

  Выражение разборВыраженияНов(/*Выражение в*/)
  {
    auto начало = сема;
    пропусти(Т.Нов);

    Выражение[] аргиНов;
    Выражение[] аргиКтора;

    if (сема.вид == Т.ЛСкобка)
      аргиНов = разборАргументов();

    // ВыражениеНовАнонКласс:
    //         new (ArgumentList)opt class (ArgumentList)opt SuperClassopt
ЦелerfaceClassesopt ClassBody
```

```
    if (проверено(Т.Класс))
    {
      if (сема.вид == Т.ЛСкобка)
        аргиКтора = разборАргументов();

      ТипКлассОснова[] основы = сема.вид != Т.ЛФСкобка ?
разборБазовыхКлассов(нет) : null ;

      auto деклы = разборТелаДефинициидекларации();
      return установи(new ВыражениеНовАнонКласс(/*в, */аргиНов, основы,
аргиКтора, деклы), начало);
    }

    // ВыражениеНов:
    //          NewArguments Тип [ ВыражениеПрисвой ]
    //          NewArguments Тип ( ArgumentList )
    //          NewArguments Тип
    auto тип = разборТипа();

    if (сема.вид == Т.ЛСкобка)
      аргиКтора = разборАргументов();

    return установи(new ВыражениеНов(/*в, */аргиНов, тип, аргиКтора),
начало);
  }

  /// Parses a Тип.
  Тип разборТипа()
  {
    return разборБазовогоТипа2(разборБазовогоТипа());
  }

  Тип разборТипаИдентификатора()
  {
    auto начало = сема;
    auto идент = требуетсяИдентификатор(сооб.ОжидалсяИдентификатор);
    Тип t;
    if (проверено(Т.Не)) // Идентификатор !( АргументыШаблона )
      t = new ТЭкземплярШаблона(идент, parseTemplateArguments());
    else // Идентификатор
      t = new ТИдентификатор(идент);
    return установи(t, начало);
  }

  Тип разборКвалифицированногоТипа()
  {
    auto начало = сема;
    Тип тип;
    if (сема.вид == Т.Точка)
      тип = установи(new ТМасштабМодуля(), начало, начало);
    else if (сема.вид == Т.Типа)
      тип = parseTypeofType();
    else
      тип = разборТипаИдентификатора();

    while (проверено(Т.Точка))
      тип = установи(new КвалифицированныйТип(тип,
разборТипаИдентификатора()), начало);
    return тип;
  }

  Тип разборБазовогоТипа()
  {
```

```
  auto начало = сема;
  Тип t;

  if (сема.интегральныйТип_ли)
  {
    t = new ИнтегральныйТип(сема.вид);
    далее();
  }
  else
  switch (сема.вид)
  {
  case Т.Идентификатор, Т.Типа, Т.Точка:
    t = разборКвалифицированногоТипа();
    return t;
  version(D2)
  {
  case Т.Конст:
    // const ( Тип )
    требуетсяСледующий(Т.ЛСкобка);
    t = разборТипа();
    требуется(Т.ПСкобка);
    t = new ТКонст(t);
    break;
  case Т.Инвариант:
    // invariant ( Тип )
    требуетсяСледующий(Т.ЛСкобка);
    t = разборТипа();
    требуется(Т.ПСкобка);
    t = new ТИнвариант(t);
    break;
  } // version(D2)
  default:
    ошибка2(ИДС.НайденоИноеЧемОжидалось, "BasicType", сема);
    t = new НелегальныйТип();
    далее();
  }
  return установи(t, начало);
}

Тип разборБазовогоТипа2(Тип t)
{
  while (1)
  {
    auto начало = сема;
    switch (сема.вид)
    {
    case Т.Умножь:
      t = new ТУказатель(t);
      далее();
      break;
    case Т.ЛКвСкобка:
      t = разборТипаМассив(t);
      continue;
    case Т.Функция, Т.Делегат:
      ТОК лекс = сема.вид;
      далее();
      auto параметры = разборСпискаПараметров();
      if (лекс == Т.Функция)
        t = new ТФункция(t, параметры);
      else
        t = new ТДелегат(t, параметры);
      break;
    default:
```

```
        return t;
      }
      установи(t, начало);
    }
    assert(0);
  }

  /// Returns да if the сема after the закрывающий parenthesis
  /// matches the searched вид.
  бул семаПослеСкобкиЯвляется(ТОК вид)
  {
    assert(сема.вид == Т.ЛСкобка);
    auto следщ = сема;
    return пропустиСкобку(следщ) == вид;
  }

  /// определено
  бул семаПослеСкобкиЯвляется(ТОК вид, ref Сема* следщ)
  {
    assert(следщ !is null && следщ.вид == Т.ЛСкобка);
    return пропустиСкобку(следщ) == вид;
  }

  /// Skips в the сема behind the закрывающий parenthesis.
  /// Takes nested parentheses into account.
  ТОК пропустиСкобку(ref Сема* следщ)
  {
    assert(следщ !is null && следщ.вид == Т.ЛСкобка);
    // We счёт nested parentheses семы because template types, typeof etc.
    // may appear внутри parameter lists. E.g.: (цел x, Foo!(цел) y)
    бцел уровень = 1;
  Loop:
    while (1)
      switch (возьмиПосле(следщ))
      {
      case Т.ЛСкобка:
        ++уровень;
        break;
      case Т.ПСкобка:
        if (--уровень == 0)
          return возьмиПосле(следщ); // Closing parenthesis found.
        break;
      case Т.КФ:
        return Т.КФ;
      default:
      }
    assert(0, "должно быть недоступно");
  }

  /// Parse the массив types after the declarator (C-style.) E.g.: цел a[]
  Тип разборСуффиксаДекларатора(Тип lhsType)
  {
    // The Тип chain should be as follows:
    // цел[3]* Идентификатор [][32]
    //   <- <-             ->  -.
    //       ^-----------------´
    // Итоging chain: [][32]*[3]цел
    Тип разборСледующего() // Nested function required в accomplish this.
    {
      if (сема.вид != Т.ЛКвСкобка)
        return lhsType; // Всё recursion; return Тип on the левый hand
сторона of the Идентификатор.
```

```
      auto начало = сема;
      Тип t;
      пропусти(Т.ЛКвСкобка);
      if (проверено(Т.ПКвСкобка))
        t = new ТМассив(разборСледующего()); // [ ]
      else
      {
        бул успех;
        Тип разборТипаАМ()
        {
          auto тип = разборТипа();
          требуется(Т.ПКвСкобка);
          return тип;
        }
        auto ассоцТип = пробуй_(&разборТипаАМ, успех);
        if (успех)
          t = new ТМассив(разборСледующего(), ассоцТип); // [ Тип ]
        else
        {
          Выражение в = разборВыражения(), е2;
          if (проверено(Т.Срез))
            е2 = разборВыражения();
          требуетсяЗакрыв(Т.ПКвСкобка, начало);
          t = new ТМассив(разборСледующего(), в, е2); // [ Выражение ..
Выражение ]
        }
      }
      установи(t, начало);
      return t;
    }
    return разборСледующего();
  }

  Тип разборТипаМассив(Тип t)
  {
    auto начало = сема;
    пропусти(Т.ЛКвСкобка);
    if (проверено(Т.ПКвСкобка))
      t = new ТМассив(t);
    else
    {
      бул успех;
      Тип разборТипаАМ()
      {
        auto тип = разборТипа();
        требуется(Т.ПКвСкобка);
        return тип;
      }
      auto ассоцТип = пробуй_(&разборТипаАМ, успех);
      if (успех)
        t = new ТМассив(t, ассоцТип);
      else
      {
        Выражение в = разборВыражения(), е2;
        if (проверено(Т.Срез))
          е2 = разборВыражения();
        требуетсяЗакрыв(Т.ПКвСкобка, начало);
        t = new ТМассив(t, в, е2);
      }
    }
    установи(t, начало);
    return t;
  }
```

```
  Тип разборТипаУказательНаФункциюСи(Тип тип, ref Идентификатор* идент, бул
опцСписокПарам)
  {
    assert(тип !is null);
    auto начало = сема;
    пропусти(Т.ЛСкобка);

    тип = разборБазовогоТипа2(тип);
    if (сема.вид == Т.ЛСкобка)
    { // Can be nested.
      тип = разборТипаУказательНаФункциюСи(тип, идент, да);
    }
    else if (сема.вид == Т.Идентификатор)
    { // The identifier of the function pointer and the declaration.
      идент = сема.идент;
      далее();
      тип = разборСуффиксаДекларатора(тип);
    }
    требуетсяЗакрыв(Т.ПСкобка, начало);

    Параметры парамы;
    if (опцСписокПарам)
      парамы = сема.вид == Т.ЛСкобка ? разборСпискаПараметров() : null;
    else
      парамы = разборСпискаПараметров();

    тип = new ТУказательНаФункСи(тип, парамы);
    return установи(тип, начало);
  }

  Тип разборДекларатора(ref Идентификатор* идент, бул identOptional = нет)
  {
    auto t = разборТипа();

    if (сема.вид == Т.ЛСкобка)
      t = разборТипаУказательНаФункциюСи(t, идент, да);
    else if (сема.вид == Т.Идентификатор)
    {
      идент = сема.идент;
      далее();
      t = разборСуффиксаДекларатора(t);
    }

    if (идент is null && !identOptional)
      ошибка2(сооб.ОжидалсяИдентификаторДекларатора, сема);

    return t;
  }

  /// Parses a список of AssignExpressions.
  /// $(PRE
  /// ExpressionList :=
  ///   ВыражениеПрисвой
  ///   ВыражениеПрисвой , ExpressionList
  /// )
  Выражение[] разборСпискаВыражений()
  {
    Выражение[] выражения;
    do
      выражения ~= разборВыраженияПрисвой();
    while(проверено(Т.Запятая))
    return выражения;
```

```
  }

  /// Parses a список of Аргументы.
  /// $(PRE
  /// Аргументы :=
  ///   ( )
  ///   ( ExpressionList )
  /// )
  Выражение[] разборАргументов()
  {
    auto leftParen = сема;
    пропусти(Т.ЛСкобка);
    Выражение[] арги;
    if (сема.вид != Т.ПСкобка)
      арги = разборСпискаВыражений();
    требуетсяЗакрыв(Т.ПСкобка, leftParen);
    return арги;
  }

  /// Parses a ParameterList.
  Параметры разборСпискаПараметров()
  out(парамы)
  {
    if (парамы.length > 1)
      foreach (парам; парамы.элементы[0..$-1])
      {
        if (парам.вариадический_ли())
          assert(0, "вариадические аргументы могут появляться только в конце
списка параметров.");
      }
  }
  body
  {
    auto начало = сема;
    требуется(Т.ЛСкобка);

    auto парамы = new Параметры();

    if (проверено(Т.ПСкобка))
      return установи(парамы, начало);

    do
    {
      auto paramBegin = сема;
      КлассХранения кхр, stc_;
      Тип тип;
      Идентификатор* идент;
      Выражение дефЗначение;

      проц  pushParameter()
      {
        парамы ~= установи(new Параметр(кхр, тип, идент, дефЗначение),
paramBegin);
      }

      if (проверено(Т.Эллипсис))
      {
        кхр = КлассХранения.Вариадический;
        pushParameter(); // тип, идент and дефЗначение will be null.
        break;
      }

      while (1)
```

```
{ // Parse storage classes.
  switch (сема.вид)
  {
version(D2)
{
  case Т.Инвариант: // D2.0
    if (возьмиСледщ() == Т.ЛСкобка)
      break;
    stc_ = КлассХранения.Инвариант;
    goto Lcommon;
  case Т.Конст: // D2.0
    if (возьмиСледщ() == Т.ЛСкобка)
      break;
    stc_ = КлассХранения.Конст;
    goto Lcommon;
  case Т.Окончательный: // D2.0
    stc_ = КлассХранения.Окончательный;
    goto Lcommon;
  case Т.Масштаб: // D2.0
    stc_ = КлассХранения.Масштаб;
    goto Lcommon;
  case Т.Статический: // D2.0
    stc_ = КлассХранения.Статический;
    goto Lcommon;
}
  case Т.Вхо:
    stc_ = КлассХранения.Вхо;
    goto Lcommon;
  case Т.Вых:
    stc_ = КлассХранения.Вых;
    goto Lcommon;
  case Т.Вховых, Т.Реф:
    stc_ = КлассХранения.Реф;
    goto Lcommon;
  case Т.Отложенный:
    stc_ = КлассХранения.Отложенный;
    goto Lcommon;
  Lcommon:
    // Check for redundancy.
    if (кхр & stc_)
      ошибка2(ИДС.ПовторяющийсяКлассХранения, сема);
    else
      кхр |= stc_;
    далее();
  version(D2)
    continue;
  else
    break; // Вхо D1.0 the grammar only allows one storage class.
  default:
  }
  break; // Всё out of inner loop.
}
тип = разборДекларатора(идент, да);

if (проверено(Т.Присвоить))
  дефЗначение = разборВыраженияПрисвой();

if (проверено(Т.Эллипсис))
{
  кхр |= КлассХранения.Вариадический;
  pushParameter();
  break;
}
```

```
      pushParameter();

    } while (проверено(Т.Запятая))
    требуетсяЗакрыв(Т.ПСкобка, начало);
    return установи(парамы, начало);
  }

  АргументыШаблона parseTemplateArguments()
  {
    АргументыШаблона шарги;
    auto leftParen = сема;
    требуется(Т.ЛСкобка);
    if (сема.вид != Т.ПСкобка)
      шарги = разборАргументовШаблона_();
    требуетсяЗакрыв(Т.ПСкобка, leftParen);
    return шарги;
  }

version(D2)
{
  АргументыШаблона parseTemplateArguments2()
  {
    пропусти(Т.Запятая);
    АргументыШаблона шарги;
    if (сема.вид != Т.ПСкобка)
      шарги = разборАргументовШаблона_();
    else
      ошибка(сема, сооб.ОжидалсяТипИлиВыражение);
    требуется(Т.ПСкобка);
    return шарги;
  }
} // version(D2)

  АргументыШаблона разборАргументовШаблона_()
  {
    auto начало = сема;
    auto шарги = new АргументыШаблона;
    do
    {
      Тип parseType_()
      {
        auto тип = разборТипа();
        if (сема.вид == Т.Запятая || сема.вид == Т.ПСкобка)
          return тип;
        провал_пробы();
        return null;
      }
      был успех;
      auto typeArgument = пробуй_(&parseType_, успех);
      if (успех)
        // TemplateArgument:
        //         Тип
        //         Символ
        шарги ~= typeArgument;
      else
        // TemplateArgument:
        //         ВыражениеПрисвой
        шарги ~= разборВыраженияПрисвой();
    } while (проверено(Т.Запятая))
    установи(шарги, начало);
    return шарги;
  }
```

```
  /// if ( ConstraintExpression )
  Выражение разборДополнительногоКонстрейнта()
  {
    if (!проверено(Т.Если))
      return null;
    требуется(Т.ЛСкобка);
    auto в = разборВыражения();
    требуется(Т.ПСкобка);
    return в;
  }

  ПараметрыШаблона разборСпискаПараметровШаблона()
  {
    auto начало = сема;
    auto шпарамы = new ПараметрыШаблона;
    требуется(Т.ЛСкобка);
    if (сема.вид != Т.ПСкобка)
      разборСпискаПараметровШаблона_(шпарамы);
    требуетсяЗакрыв(Т.ПСкобка, начало);
    return установи(шпарамы, начало);
  }

version(D2)
{
  ПараметрыШаблона разборСпискаПараметровШаблона2()
  {
    пропусти(Т.Запятая);
    auto начало = сема;
    auto шпарамы = new ПараметрыШаблона;
    if (сема.вид != Т.ПСкобка)
      разборСпискаПараметровШаблона_(шпарамы);
    else
      ошибка(сема, сооб.ОжидалисьПараметрыШаблона);
    return установи(шпарамы, начало);
  }
} // version(D2)

  /// Parses template параметры.
  проц  разборСпискаПараметровШаблона_(ПараметрыШаблона шпарамы)
  {
    do
    {
      auto paramBegin = сема;
      ПараметрШаблона тр;
      Идентификатор* идент;
      Тип типСпец, дефТип;

      проц  разборСпецИИлиДефолтнТипа()
      {
        // : SpecializationType
        if (проверено(Т.Двоеточие))
          типСпец = разборТипа();
        // = DefaultType
        if (проверено(Т.Присвоить))
          дефТип = разборТипа();
      }

      switch (сема.вид)
      {
      case Т.Алиас:
        // ПараметрАлиасШаблона:
        //         alias Идентификатор
        пропусти(Т.Алиас);
```

```
        идент = требуетсяИдентификатор(сооб.ОжидалсяАлиасПараметраШаблона);
        разборСпецИИлиДефолтнТипа();
        тр = new ПараметрАлиасШаблона(идент, типСпец, дефТип);
        break;
      case Т.Идентификатор:
        идент = сема.идент;
        switch (возьмиСледщ())
        {
        case Т.Эллипсис:
          // ПараметрКортежШаблона:
          //         Идентификатор ...
          пропусти(Т.Идентификатор); пропусти(Т.Эллипсис);
          if (сема.вид == Т.Запятая)
            ошибка(ИДС.ПараметрКортежШаблона);
          тр = new ПараметрКортежШаблона(идент);
          break;
        case Т.Запятая, Т.ПСкобка, Т.Двоеточие, Т.Присвоить:
          // ПараметрТипаШаблона:
          //         Идентификатор
          пропусти(Т.Идентификатор);
          разборСпецИИлиДефолтнТипа();
          тр = new ПараметрТипаШаблона(идент, типСпец, дефТип);
          break;
        default:
          // ПараметрШаблонЗначения:
          //         Declarator
          идент = null;
          goto LTemplateValueParameter;
        }
        break;
      version(D2)
      {
      case Т.Этот:
        // ПараметрЭтотШаблона
        //         this ПараметрТипаШаблона
        пропусти(Т.Этот);
        идент =
требуетсяИдентификатор(сооб.ОжидалосьИмяДляПараметраШаблонаThis);
        разборСпецИИлиДефолтнТипа();
        тр = new ПараметрЭтотШаблона(идент, типСпец, дефТип);
        break;
      }
      default:
      LTemplateValueParameter:
        // ПараметрШаблонЗначения:
        //         Declarator
        Выражение спецЗначение, дефЗначение;
        auto типЗначение = разборДекларатора(идент);
        // : SpecializationValue
        if (проверено(Т.Двоеточие))
          спецЗначение = разборУсловнВыражения();
        // = DefaultValue
        if (проверено(Т.Присвоить))
          дефЗначение = разборУсловнВыражения();
        тр = new ПараметрШаблонЗначения(типЗначение, идент, спецЗначение,
дефЗначение);
      }

      // Push template parameter.
      шпарамы ~= установи(тр, paramBegin);

    } while (проверено(Т.Запятая))
  }
```

```
/// Возвращает ткст of a сема printable в the client.
ткст дайПечатный(Сема* сема)
{ // TODO: there are some другой семы that have в be handled, в.g. тксты.
  return сема.вид == Т.КФ ? "КФ" : сема.исхТекст;
}

alias требуется ожидаемое;

/// Requires a сема of вид лекс.
проц  требуется(ТОК лекс)
{
  if (сема.вид == лекс)
    далее();
  else
    ошибка2(ИДС.НайденоИноеЧемОжидалось, Сема.вТкст(лекс), сема);
}

/// Requires the следщ сема в be of вид лекс.
проц  требуетсяСледующий(ТОК лекс)
{
  далее();
  требуется(лекс);
}

/// Optionally parses an identifier.
/// Возвращает: null or the identifier.
Идентификатор* дополнительныйИдентификатор()
{
  Идентификатор* ид;
  if (сема.вид == Т.Идентификатор)
    (ид = сема.идент), пропусти(Т.Идентификатор);
  return ид;
}

Идентификатор* требуетсяИдентификатор()
{
  Идентификатор* ид;
  if (сема.вид == Т.Идентификатор)
    (ид = сема.идент), пропусти(Т.Идентификатор);
  else
    ошибка(ИДС.НайденоИноеЧемОжидалось, "Идентификатор", сема.исхТекст);
  return ид;
}

/// Reports an ошибка if the current сема is not an identifier.
/// Параметры:
///   ошСооб = the ошибка сообщение в be used.
/// Возвращает: null or the identifier.
Идентификатор* требуетсяИдентификатор(ткст ошСооб)
{
  Идентификатор* ид;
  if (сема.вид == Т.Идентификатор)
    (ид = сема.идент), пропусти(Т.Идентификатор);
  else
    ошибка(сема, ошСооб, сема.исхТекст);
  return ид;
}

/// Reports an ошибка if the current сема is not an identifier.
/// Параметры:
///   идс = the ошибка сообщение ID в be used.
/// Возвращает: null or the identifier.
```

```
  Идентификатор* требуетсяИдентификатор(ИДС идс)
  {
    Идентификатор* ид;
    if (сема.вид == Т.Идентификатор)
      (ид = сема.идент), пропусти(Т.Идентификатор);
    else
      ошибка(идс, сема.исхТекст);
    return ид;
  }

  /// Reports an ошибка if the current сема is not an identifier.
  /// Возвращает: null or the сема.
  Сема* требуетсяИд()
  {
    Сема* семаИд;
    if (сема.вид == Т.Идентификатор)
      (семаИд = сема), пропусти(Т.Идентификатор);
    else
      ошибка(ИДС.НайденоИноеЧемОжидалось, "Идентификатор", сема.исхТекст);
    return семаИд;
  }

  Сема* требуетсяСемаИд(ткст ошСооб)
  {
    Сема* семаИд;
    if (сема.вид == Т.Идентификатор)
      (семаИд = сема), пропусти(Т.Идентификатор);
    else
    {
      ошибка(сема, ошСооб, сема.исхТекст);
      семаИд = лексер.вставьПустуюСемуПеред(сема);
      this.предыдущСема = семаИд;
    }
    return семаИд;
  }

  /// Reports an ошибка if the закрывающий counterpart of a сема is not
found.
  проц  требуетсяЗакрыв(ТОК закрывающий, Сема* открывающий)
  {
    assert(закрывающий == Т.ПФСкобка || закрывающий == Т.ПСкобка ||
закрывающий == Т.ПКвСкобка);
    assert(открывающий !is null);
    if (!проверено(закрывающий))
    {
      auto место = открывающий.дайРеальноеПоложение();
      auto открывающийLoc = Формат("(открывающий @{},{})", место.номСтр,
место.номСтолб);
      //ошибка(сема, сооб.ExpectedClosing,
            //Сема.вТкст(закрывающий), открывающийLoc, дайПечатный(сема));
    }
  }

  /// Returns да if the ткст ткт has an invalid UTF-8 sequence.
  бул естьНеверныйЮ8(ткст ткт, Сема* начало)
  {
    auto invalidUTF8Seq =
Лексер.найдиНедействительнуюПоследовательностьУТФ8(ткт);
    if (invalidUTF8Seq.length)
      ошибка(начало, сооб.НедействительнаяПоследовательностьУТФ8ВТексте,
invalidUTF8Seq);
    return invalidUTF8Seq.length != 0;
  }
```

```
  /// Forwards ошибка параметры.
  проц  ошибка(Сема* сема, ткст форматирСооб, ...)
  {
    error_(сема, форматирСооб, _arguments, _argptr);
  }
  /// определено
  проц  ошибка(ИДС идс, ...)
  {
    error_(this.сема, ДайСооб(идс), _arguments, _argptr);
  }

  /// определено
  проц  ошибка2(ткст форматирСооб, Сема* сема)
  {
    ошибка(сема, форматирСооб, дайПечатный(сема));
  }
  /// определено
  проц  ошибка2(ИДС идс, Сема* сема)
  {
    ошибка(идс, дайПечатный(сема));
  }
  /// определено
  проц  ошибка2(ИДС идс, ткст арг, Сема* сема)
  {
    ошибка(идс, арг, дайПечатный(сема));
  }

  /// Создаёт отчёт об ошибках и добавляет его в список.
  /// Параметры:
  ///   сема = используется для получения позиции ошибки.
  ///   форматирСооб = сообщение компилятора об ошибке.
  проц  error_(Сема* сема, ткст форматирСооб, TypeInfo[] _arguments,
base.спис_ва _argptr)
  {
    if (пробуем)
    {
      счётОшибок++;
      return;
    }
    auto положение = сема.дайПоложениеОшибки();
    auto сооб = Формат(_arguments, _argptr, форматирСооб);
    auto ошибка = new ОшибкаПарсера(положение, сооб);
    ошибки ~= ошибка;
    if (диаг !is null)
      диаг ~= ошибка;
  }
}
```