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 ДефинцияДекларации.
/// 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 вырРеле;
  this(Идентификатор* идент, Выражение вырРеле)
  {
    mixin(установить_вид);
    добавьОпцОтпрыск(вырРеле);
    this.идент = идент;
    this.вырРеле = вырРеле;
  }
  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;
  }

  T 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;
  }

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

  T 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;
  }

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

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

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

  T 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 findOverload(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 findOverload(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 = findOverload(в, Идент.opOr, Идент.opOr_r))
      return о;
    return в;
  }

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

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

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

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

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

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

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

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

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

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

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

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

В посети(ВыражениеУмножь в)
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

В посети(ВыражениеПрисвойСоед в)
{
  auto о = findOverload(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 = findOverload(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 = findOverload(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 = findOverload(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 = findOverload(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 = findOverload(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 = findOverload(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 = findOverload(в, Идент.opCast))
    return о;
  return в;
}

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

В посети(ВыражениеСрез в)
{
  if (auto о = findOverload(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(t.лв);
    // Присвоить the символ of the левый-hand сторона в идМасштаб.
    setIdScope(t.лв.символ);
    посетиТ(t.пв);
//      setIdScope(t.пв.символ);
    // Присвоить члены of the правый-hand сторона в this тип.
    t.тип = t.пв.тип;
    t.символ = t.пв.символ;
    // Reset идМасштаб.
    resetIdScope && (идМасштаб = null);
    return t;
  }

  Т посети(ТМасштабМодуля t)
  {
    идМасштаб = модуль;
    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(ТТип t)
  {
    t.в = посетиВ(t.в);
    t.тип = t.в.тип;
    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(ТУказатель t)
  {
    t.тип = посетиТ(t.следщ).тип.укНа();
    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(ТФункция t)
```

```
  {
    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
}
```