**Проект DRC**

**Разработчик Dinrus Group,**

**Виталий Кулич**

[**http://github.com/DinrusGroup/DRC**](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** "переп"**;**

**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**.**C**:** **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**.**C**:** **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 ДефиницияДекларации.

/// 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 ДекларацииШаблона.

//

**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 identеслиier.

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 **==** TOK**.**Плюс**;**

**}**

бул отриц\_ли**()**

**{**

**assert(**начало **!is** **null);**

**return** начало**.**вид **==** TOK**.**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 **==** TOK**.**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 TOK.Цел32:

// тип = Типы.Цел; break;

**case** TOK**.**Бцел32**:**

тип **=** Типы**.**Бцел**;** **break;**

**case** TOK**.**Цел64**:**

тип **=** Типы**.**Дол**;** **break;**

**case** TOK**.**Бцел64**:**

тип **=** Типы**.**Бдол**;** **break;**

**default:**

**assert(**сема**.**вид **==** TOK**.**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** TOK**.**Плав32**:**

тип **=** Типы**.**Плав**;** **break;**

// case TOK.Плав64:

// тип = Типы.Дво; break;

**case** TOK**.**Плав80**:**

тип **=** Типы**.**Реал**;** **break;**

**case** TOK**.**Мнимое32**:**

тип **=** Типы**.**Вплав**;** **break;**

**case** TOK**.**Мнимое64**:**

тип **=** Типы**.**Вдво**;** **break;**

**case** TOK**.**Мнимое80**:**

тип **=** Типы**.**Вреал**;** **break;**

**default:**

**assert(**сема**.**вид **==** TOK**.**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"**,** "e2"**];**

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**]** **=** **[**"вырРеле?"**];**

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?"**,** "e1?"**,** "e2?"**];**

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

**{**

TOK лекс**;**

Параметры парамы**;**

Выражение агрегат**;**

Инструкция телоПри**;**

**this(**TOK 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 **==** TOK**.**Длявсех\_реверс**;**

**}**

**mixin(**методКопирования**);**

**}**

// version(D2)

// {

**class** **ИнструкцияДиапазонСКаждым** **:** Инструкция

**{**

TOK лекс**;**

Параметры парамы**;**

Выражение нижний**,** верхний**;**

Инструкция телоПри**;**

**this(**TOK 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

**{**

TOK лекс**;**

**this(**TOK 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\_ a "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 **==** TOK**.**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 **==** TOK**.**Идентификатор **||** услов**.**вид **==** TOK**.**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 символвов 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**.**иниты**[**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 **==** TOK**.**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 **==** TOK**.**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 **==** TOK**.**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 a 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**[**TOK**]** семВТип **=** **[**

TOK**.**Сим **:** Типы**.**Сим**,** TOK**.**Шим **:** Типы**.**Шим**,** TOK**.**Дим **:** Типы**.**Дим**,** TOK**.**Бул **:** Типы**.**Бул**,**

TOK**.**Байт **:** Типы**.**Байт**,** TOK**.**Ббайт **:** Типы**.**Ббайт**,** TOK**.**Крат **:** Типы**.**Крат**,** TOK**.**Бкрат **:** Типы**.**Бкрат**,**

TOK**.**Цел **:** Типы**.**Цел**,** TOK**.**Бцел **:** Типы**.**Бцел**,** TOK**.**Дол **:** Типы**.**Дол**,** TOK**.**Бдол **:** Типы**.**Бдол**,**

TOK**.**Цент **:** Типы**.**Цент**,** TOK**.**Бцент **:** Типы**.**Бцент**,**

TOK**.**Плав **:** Типы**.**Плав**,** TOK**.**Дво **:** Типы**.**Дво**,** TOK**.**Реал **:** Типы**.**Реал**,**

TOK**.**Вплав **:** Типы**.**Вплав**,** TOK**.**Вдво **:** Типы**.**Вдво**,** TOK**.**Вреал **:** Типы**.**Вреал**,**

TOK**.**Кплав **:** Типы**.**Кплав**,** TOK**.**Кдво **:** Типы**.**Кдво**,** TOK**.**Креал **:** Типы**.**Креал**,** TOK**.**Проц **:** Типы**.**Проц

**];**

**assert(**t**.**лекс **in** семВТип**);**

t**.**тип **=** семВТип**[**t**.**лекс**];**

**return** t**;**

**}**

// Выражение nodes:

E посети**(**ВыражениеРодит в**)**

**{**

**if** **(!**в**.**тип**)**

**{**

в**.**следщ **=** посетиВ**(**в**.**следщ**);**

в**.**тип **=** в**.**следщ**.**тип**;**

**}**

**return** в**;**

**}**

E посети**(**ВыражениеЗапятая в**)**

**{**

**if** **(!**в**.**тип**)**

**{**

в**.**лв **=** посетиВ**(**в**.**лв**);**

в**.**пв **=** посетиВ**(**в**.**пв**);**

в**.**тип **=** в**.**пв**.**тип**;**

**}**

**return** в**;**

**}**

E посети**(**ВыражениеИлиИли**)**

**{** **return** **null;** **}**

E посети**(**ВыражениеИИ**)**

**{** **return** **null;** **}**

E посети**(**ВыражениеСпецСема в**)**

**{**

**if** **(**в**.**тип**)**

**return** в**.**значение**;**

**switch** **(**в**.**особаяСема**.**вид**)**

**{**

**case** TOK**.**СТРОКА**,** TOK**.**ВЕРСИЯ**:**

в**.**значение **=** **new** ЦелВыражение**(**в**.**особаяСема**.**бцел\_**,** Типы**.**Бцел**);**

**break;**

**case** TOK**.**ФАЙЛ**,** TOK**.**ДАТА**,** TOK**.**ВРЕМЯ**,** TOK**.**ШТАМПВРЕМЕНИ**,** TOK**.**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 пропустиped during a символ ищи.

Символ ignoreSymbol**;**

/// The current Масштаб символ в use for looking up identifiers.

/// В.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 **==** TOK**.**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**.**иниты**[**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 **==** TOK**.**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 **==** TOK**.**Идентификатор**)**

контекст**.**добавьИдВерсии**(**d**.**спец**.**идент**.**ткт**);**

**else**

контекст**.**уровеньВерсии **=** d**.**спец**.**бцел\_**;**

**}**

**else**

**{** // version ( Condition )

**if** **(**выборВерсионВетви**(**d**.**услов**,** контекст**))**

d**.**компилированныеДеклы **=** d**.**деклы**;**

**else**

d**.**компилированныеДеклы **=** d**.**деклыИначе**;**

d**.**компилированныеДеклы **&&** посетиД**(**d**.**компилированныеДеклы**);**

**}**

**return** d**;**

**}**

Д посети**(**ДекларацияШаблона d**)**

**{**

**if** **(**d**.**символ**)**

**return** d**;**

// Create the символ.

d**.**символ **=** **new** Шаблон**(**d**.**имя**,** d**);**

// Insert into current Масштаб.

вставьПерегрузку**(**d**.**символ**);**

**return** d**;**

**}**

Д посети**(**ДекларацияНов d**)**

**{**

**auto** func **=** **new** Функция**(**Идент**.**Нов**,** d**);**

вставь**(**func**);**

**return** d**;**

**}**

Д посети**(**ДекларацияУдали d**)**

**{**

**auto** func **=** **new** Функция**(**Идент**.**Удалить**,** d**);**

вставь**(**func**);**

**return** d**;**

**}**

// Attributes:

Д посети**(**ДекларацияЗащиты d**)**

**{**

**auto** saved **=** защита**;** // Save.

защита **=** d**.**защ**;** // Set.

посетиД**(**d**.**деклы**);**

защита **=** saved**;** // Restore.

**return** d**;**

**}**

Д посети**(**ДекларацияКлассаХранения d**)**

**{**

**auto** saved **=** классХранения**;** // Save.

классХранения **=** d**.**классХранения**;** // Set.

посетиД**(**d**.**деклы**);**

классХранения **=** saved**;** // Restore.

**return** d**;**

**}**

Д посети**(**ДекларацияКомпоновки d**)**

**{**

**auto** saved **=** типКомпоновки**;** // Save.

типКомпоновки **=** d**.**типКомпоновки**;** // Set.

посетиД**(**d**.**деклы**);**

типКомпоновки **=** saved**;** // Restore.

**return** d**;**

**}**

Д посети**(**ДекларацияРазложи d**)**

**{**

**auto** saved **=** размерРаскладки**;** // Save.

размерРаскладки **=** d**.**размер**;** // Set.

посетиД**(**d**.**деклы**);**

размерРаскладки **=** saved**;** // Restore.

**return** d**;**

**}**

Д посети**(**ДекларацияСтатическогоПодтверди d**)**

**{**

**return** d**;**

**}**

Д посети**(**ДекларацияСтатическогоЕсли d**)**

**{**

**return** d**;**

**}**

Д посети**(**ДекларацияСмеси d**)**

**{**

**return** d**;**

**}**

Д посети**(**ДекларацияПрагмы d**)**

**{**

**if** **(**d**.**идент **is** Идент**.**сооб**)**

**{**

// TODO

**}**

**else**

**{**

семантикаПрагмы**(**масш**,** d**.**начало**,** d**.**идент**,** d**.**арги**);**

посетиД**(**d**.**деклы**);**

**}**

**return** d**;**

**}**

**}** // override

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

| Statements |

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

/// The current surrounding, breakable statement.

И breakableStatement**;**

И setBS**(**И s**)**

**{**

**auto** old **=** breakableStatement**;**

breakableStatement **=** s**;**

**return** old**;**

**}**

проц restoreBS**(**И s**)**

**{**

breakableStatement **=** s**;**

**}**

**override**

**{**

И посети**(**СложнаяИнструкция s**)**

**{**

**foreach** **(**stmnt**;** s**.**инстрции**)**

посетиИ**(**stmnt**);**

**return** s**;**

**}**

И посети**(**НелегальнаяИнструкция**)**

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

И посети**(**ПустаяИнструкция s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияТелаФункции s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияМасштаб s**)**

**{**

// войдиВМасштаб();

посетиИ**(**s**.**s**);**

// выйдиИзМасштаба();

**return** s**;**

**}**

И посети**(**ИнструкцияСМеткой s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияВыражение s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияДекларация s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияЕсли s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияПока s**)**

**{**

**auto** saved **=** setBS**(**s**);**

// TODO:

restoreBS**(**saved**);**

**return** s**;**

**}**

И посети**(**ИнструкцияДелайПока s**)**

**{**

**auto** saved **=** setBS**(**s**);**

// TODO:

restoreBS**(**saved**);**

**return** s**;**

**}**

И посети**(**ИнструкцияПри s**)**

**{**

**auto** saved **=** setBS**(**s**);**

// TODO:

restoreBS**(**saved**);**

**return** s**;**

**}**

И посети**(**ИнструкцияСКаждым s**)**

**{**

**auto** saved **=** setBS**(**s**);**

// TODO:

// find overload opApply or opApplyReverse.

restoreBS**(**saved**);**

**return** s**;**

**}**

// D2.0

И посети**(**ИнструкцияДиапазонСКаждым s**)**

**{**

**auto** saved **=** setBS**(**s**);**

// TODO:

restoreBS**(**saved**);**

**return** s**;**

**}**

И посети**(**ИнструкцияЩит s**)**

**{**

**auto** saved **=** setBS**(**s**);**

// TODO:

restoreBS**(**saved**);**

**return** s**;**

**}**

И посети**(**ИнструкцияРеле s**)**

**{**

**auto** saved **=** setBS**(**s**);**

// TODO:

restoreBS**(**saved**);**

**return** s**;**

**}**

И посети**(**ИнструкцияДефолт s**)**

**{**

**auto** saved **=** setBS**(**s**);**

// TODO:

restoreBS**(**saved**);**

**return** s**;**

**}**

И посети**(**ИнструкцияДалее s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияВсё s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияИтог s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияПереход s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияДля s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияСинхр s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияПробуй s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияЛови s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияИтожь s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияСтражМасштаба s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияБрось s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияЛетучее s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияБлокАсм s**)**

**{**

**foreach** **(**stmnt**;** s**.**инструкции**.**инстрции**)**

посетиИ**(**stmnt**);**

**return** s**;**

**}**

И посети**(**ИнструкцияАсм s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияАсмРасклад s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияНелегальныйАсм**)**

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

И посети**(**ИнструкцияПрагма s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияСмесь s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияСтатическоеЕсли s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияСтатическоеПодтверди s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияОтладка s**)**

**{**

**return** s**;**

**}**

И посети**(**ИнструкцияВерсия s**)**

**{**

**return** s**;**

**}**

**}** // override

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

| Expressions |

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

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

бул errorOnUndefinedSymbol**;**

//бул errorOnUnknownSymbol;

/// Reports an ошибка if 'e' is of тип бул.

проц errorЕслиBool**(**Выражение в**)**

**{**

ошибка**(**в**.**начало**,** "the operation is not defined for the тип бул"**);**

**}**

/// Returns a call выражение if 'e' overrides

/// an operatorwith the имя 'ид'.

/// Параметры:

/// в = the binary выражение в be checked.

/// ид = the имя of the overload function.

Выражение найдиПерегрузку**(**УнарноеВыражение в**,** Идентификатор**\*** ид**)**

**{**

// TODO:

// check в for struct or class

// ищи for function named ид

// return call выражение: в.opXYZ()

**return** **null;**

**}**

/// Returns a call выражение if 'e' overrides

/// an operator with the имя 'ид' or 'id\_r'.

/// Параметры:

/// в = the binary выражение в be checked.

/// ид = the имя of the overload function.

/// id\_r = the имя of the reverse overload function.

Выражение найдиПерегрузку**(**БинарноеВыражение в**,** Идентификатор**\*** ид**,** Идентификатор**\*** id\_r**)**

**{**

// TODO:

**return** **null;**

**}**

**override**

**{**

В посети**(**НелегальноеВыражение**)**

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

В посети**(**ВыражениеУсловия в**)**

**{**

**return** в**;**

**}**

В посети**(**ВыражениеЗапятая в**)**

**{**

**if** **(!**в**.**естьТип**)**

**{**

в**.**лв **=** посетиВ**(**в**.**лв**);**

в**.**пв **=** посетиВ**(**в**.**пв**);**

в**.**тип **=** в**.**пв**.**тип**;**

**}**

**return** в**;**

**}**

В посети**(**ВыражениеИлиИли в**)**

**{**

**return** в**;**

**}**

В посети**(**ВыражениеИИ в**)**

**{**

**return** в**;**

**}**

В посети**(**ВыражениеИли в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opOr**,** Идент**.**opOr\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеИИли в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opXor**,** Идент**.**opXor\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеИ в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opAnd**,** Идент**.**opAnd\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеРавно в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opEquals**,** **null))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеРавенство в**)**

**{**

**return** в**;**

**}**

В посети**(**ВыражениеОтнош в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opCmp**,** **null))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеВхо в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opIn**,** Идент**.**opIn\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеЛСдвиг в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opShl**,** Идент**.**opShl\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПСдвиг в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opShr**,** Идент**.**opShr\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеБПСдвиг в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opUShr**,** Идент**.**opUShr\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПлюс в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opAdd**,** Идент**.**opAdd\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеМинус в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opSub**,** Идент**.**opSub\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеСоедини в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opCat**,** Идент**.**opCat\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеУмножь в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opMul**,** Идент**.**opMul\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеДели в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opDiv**,** Идент**.**opDiv\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеМод в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opMod**,** Идент**.**opMod\_r**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвой в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opAssign**,** **null))**

**return** o**;**

// TODO: also check for opIndexAssign and opSliceAssign.

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойЛСдвиг в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opShlAssign**,** **null))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойПСдвиг в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opShrAssign**,** **null))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойБПСдвиг в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opUShrAssign**,** **null))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойИли в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opOrAssign**,** **null))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойИ в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opAndAssign**,** **null))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойПлюс в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opAddAssign**,** **null))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойМинус в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opSubAssign**,** **null))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойДел в**)**

**{**

**auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opDivAssign**,** **null);**

**if** **(**o**)**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойУмн в**)**

**{**

**auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opMulAssign**,** **null);**

**if** **(**o**)**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойМод в**)**

**{**

**auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opModAssign**,** **null);**

**if** **(**o**)**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойИИли в**)**

**{**

**auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opXorAssign**,** **null);**

**if** **(**o**)**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеПрисвойСоед в**)**

**{**

**auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opCatAssign**,** **null);**

**if** **(**o**)**

**return** o**;**

**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** o **=** найдиПерегрузку**(**в**,** Идент**.**opPostInc**))**

**return** o**;**

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** o **=** найдиПерегрузку**(**в**,** Идент**.**opPostDec**))**

**return** o**;**

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** o **=** найдиПерегрузку**(**в**,** Идент**.**opStar**))**

**return** o**;**

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** o **=** найдиПерегрузку**(**в**,** в**.**отриц\_ли **?** Идент**.**opNeg **:** Идент**.**opPos**))**

**return** o**;**

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** o **=** найдиПерегрузку**(**в**,** Идент**.**opCom**))**

**return** o**;**

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** o **=** найдиПерегрузку**(**в**,** Идент**.**opCall**))**

**return** o**;**

**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** o **=** найдиПерегрузку**(**в**,** Идент**.**opCast**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеИндекс в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opIndex**))**

**return** o**;**

**return** в**;**

**}**

В посети**(**ВыражениеСрез в**)**

**{**

**if** **(auto** o **=** найдиПерегрузку**(**в**,** Идент**.**opSlice**))**

**return** o**;**

**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** TOK**.**СТРОКА**,** TOK**.**ВЕРСИЯ**:**

в**.**значение **=** **new** ЦелВыражение**(**в**.**особаяСема**.**бцел\_**,** Типы**.**Бцел**);**

**break;**

**case** TOK**.**ФАЙЛ**,** TOK**.**ДАТА**,** TOK**.**ВРЕМЯ**,** TOK**.**ШТАМПВРЕМЕНИ**,** TOK**.**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 t**)**

**{**

// A таблица mapping the вид of a сема в its corresponding semantic Тип.

ТипБазовый**[**TOK**]** семВТип **=** **[**

TOK**.**Сим **:** Типы**.**Сим**,** TOK**.**Шим **:** Типы**.**Шим**,** TOK**.**Дим **:** Типы**.**Дим**,** TOK**.**Бул **:** Типы**.**Бул**,**

TOK**.**Байт **:** Типы**.**Байт**,** TOK**.**Ббайт **:** Типы**.**Ббайт**,** TOK**.**Крат **:** Типы**.**Крат**,** TOK**.**Бкрат **:** Типы**.**Бкрат**,**

TOK**.**Цел **:** Типы**.**Цел**,** TOK**.**Бцел **:** Типы**.**Бцел**,** TOK**.**Дол **:** Типы**.**Дол**,** TOK**.**Бдол **:** Типы**.**Бдол**,**

TOK**.**Цент **:** Типы**.**Цент**,** TOK**.**Бцент **:** Типы**.**Бцент**,**

TOK**.**Плав **:** Типы**.**Плав**,** TOK**.**Дво **:** Типы**.**Дво**,** TOK**.**Реал **:** Типы**.**Реал**,**

TOK**.**Вплав **:** Типы**.**Вплав**,** TOK**.**Вдво **:** Типы**.**Вдво**,** TOK**.**Вреал **:** Типы**.**Вреал**,**

TOK**.**Кплав **:** Типы**.**Кплав**,** TOK**.**Кдво **:** Типы**.**Кдво**,** TOK**.**Креал **:** Типы**.**Креал**,** TOK**.**Проц **:** Типы**.**Проц

**];**

**assert(**t**.**лекс **in** семВТип**);**

t**.**тип **=** семВТип**[**t**.**лекс**];**

**return** t**;**

**}**

Т посети**(**КвалифицированныйТип t**)**

**{**

// Reset идМасштаб at the конец if this the корень КвалифицированныйТип.

бул resetIdScope **=** идМасштаб **is** **null;**

// if (t.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**;**

**}**

Т посети**(**ТДелегат t**)**

**{**

**return** t**;**

**}**

Т посети**(**ТУказательНаФункСи t**)**

**{**

**return** t**;**

**}**

Т посети**(**ТипКлассОснова t**)**

**{**

**return** t**;**

**}**

Т посети**(**ТКонст t**)** // D2.0

**{**

**return** t**;**

**}**

Т 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 p**)**

**{**

**return** p**;**

**}**

У посети**(**ПараметрКортежШаблона p**)**

**{**

**return** p**;**

**}**

У посети**(**ПараметрыШаблона p**)**

**{**

**return** p**;**

**}**

У посети**(**АргументыШаблона p**)**

**{**

**return** p**;**

**}**

**}** // 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 retriоцени 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** **ТСтруктура** **:** Тип

**{**

**this(**Символ символ**)**

**{**

**super(null,** ТИП**.**Структура**);**

**this.**символ **=** символ**;**

**}**

ткст вТкст**()**

**{**

**return** символ**.**имя**.**ткт**;**

**}**

**}**

/// Класс тип.

**class** **ТКласс** **:** Тип

**{**

**this(**Символ символ**)**

**{**

**super(null,** ТИП**.**Класс**);**

**this.**символ **=** символ**;**

**}**

ткст вТкст**()**

**{**

**return** символ**.**имя**.**ткт**;**

**}**

**}**

/// Типдеф тип.

**class** **ТипдефТип** **:** Тип

**{**

**this(**Тип следщ**)**

**{**

**super(**следщ**,** ТИП**.**Типдеф**);**

**}**

ткст вТкст**()**

**{** // TODO:

**return** "типдеф"**;**

**}**

**}**

/// Функция тип.

**class** **ФункцияТип** **:** Тип

**{**

**this(**Тип следщ**)**

**{**

**super(**следщ**,** ТИП**.**Функция**);**

**}**

ткст вТкст**()**

**{** // TODO:

**return** "функция"**;**

**}**

**}**

/// Делегат тип.

**class** **ДелегатТип** **:** Тип

**{**

**this(**Тип следщ**)**

**{**

**super(**следщ**,** ТИП**.**Делегат**);**

**}**

ткст вТкст**()**

**{** // TODO:

**return** "делегат"**;**

**}**

**}**

/// Идентификатор тип.

**class** **ИдентификаторТип** **:** Тип

**{**

Идентификатор**\*** идент**;**

**this(**Идентификатор**\*** идент**)**

**{**

**super(null,** ТИП**.**Идентификатор**);**

**}**

ткст вТкст**()**

**{**

**return** идент**.**ткт**;**

**}**

**}**

/// Шаблон instantiation тип.

**class** **ЭкземплШаблонаТип** **:** Тип

**{**

**this()**

**{**

**super(null,** ТИП**.**ШЭкземпляр**);**

**}**

ткст вТкст**()**

**{** // TODO:

**return** "шабл!()"**;**

**}**

**}**

/// Шаблон tuple тип.

**class** **КортежТип** **:** Тип

**{**

**this(**Тип следщ**)**

**{**

**super(**следщ**,** ТИП**.**Кортеж**);**

**}**

ткст вТкст**()**

**{** // TODO:

**return** "кортеж"**;**

**}**

**}**

/// Constant тип. D2.0

**class** **КонстантаТип** **:** Тип

**{**

**this(**Тип следщ**)**

**{**

**super(**следщ**,** ТИП**.**Конст**);**

**}**

ткст вТкст**()**

**{**

**return** "конст(" **~** следщ**.**вТкст**()** **~** ")"**;**

**}**

**}**

/// Инвариант тип. D2.0

**class** **ИнвариантТип** **:** Тип

**{**

**this(**Тип следщ**)**

**{**

**super(**следщ**,** ТИП**.**Конст**);**

**}**

ткст вТкст**()**

**{**

**return** "инвариант(" **~** следщ**.**вТкст**()** **~** ")"**;**

**}**

**}**

/// Represents a значение related в a Тип.

**union** **Значение**

**{**

ук упроц**;**

бул бул\_**;**

дим дим\_**;**

дол дол\_**;**

бдол бдол\_**;**

цел цел\_**;**

бцел бцел\_**;**

плав плав\_**;**

дво дво\_**;**

реал реал\_**;**

креал креал\_**;**

**}**

/// Информация related в a Тип.

**struct** **МетаИнфоТип**

**{**

сим mangle**;** /// Mangle символ of the тип.

бкрат размер**;** /// Байт размер of the тип.

Значение**\*** дефолтИниц**;** /// Дефолт initialization значение.

**}**

/// Namespace for the meta инфо таблица.

**struct** **МИТаблица**

**{**

**static:**

**const** бкрат РАЗМЕР\_НЕ\_ДОСТУПЕН **=** 0**;** /// Size not available.

**const** Значение ЗНОЛЬ **=** **{**цел\_**:**0**};** /// Значение 0.

**const** Значение ЗНУЛЬ **=** **{**упроц**:null};** /// Значение null.

**const** Значение V0xFF **=** **{**дим\_**:**0xFF**};** /// Значение 0xFF.

**const** Значение V0xFFFF **=** **{**дим\_**:**0xFFFF**};** /// Значение 0xFFFF.

**const** Значение ЗЛОЖЬ **=** **{**бул\_**:**нет**};** /// Значение нет.

**const** Значение ЗНЕЧ **=** **{**плав\_**:**плав**.**nan**};** /// Значение NAN.

**const** Значение ЗКНЕЧ **=** **{**креал\_**:**креал**.**nan**};** /// Значение complex NAN.

**private** **alias** РАЗМЕР\_НЕ\_ДОСТУПЕН РНД**;**

**private** **alias** РАЗМЕР\_УК РА**;**

/// The meta инфо таблица.

**private** **const** МетаИнфоТип метаИнфоТаблица**[]** **=** **[**

**{**'?'**,** РНД**},** // Ошибка

**{**'a'**,** 1**,** **&**V0xFF**},** // Сим

**{**'u'**,** 2**,** **&**V0xFFFF**},** // Шим

**{**'w'**,** 4**,** **&**V0xFFFF**},** // Дим

**{**'b'**,** 1**,** **&**ЗЛОЖЬ**},** // Бул

**{**'g'**,** 1**,** **&**ЗНОЛЬ**},** // Байт

**{**'h'**,** 1**,** **&**ЗНОЛЬ**},** // Ббайт

**{**'s'**,** 2**,** **&**ЗНОЛЬ**},** // Крат

**{**'t'**,** 2**,** **&**ЗНОЛЬ**},** // Бкрат

**{**'i'**,** 4**,** **&**ЗНОЛЬ**},** // Цел

**{**'k'**,** 4**,** **&**ЗНОЛЬ**},** // Бцел

**{**'l'**,** 8**,** **&**ЗНОЛЬ**},** // Дол

**{**'m'**,** 8**,** **&**ЗНОЛЬ**},** // Бдол

**{**'?'**,** 16**,** **&**ЗНОЛЬ**},** // Цент

**{**'?'**,** 16**,** **&**ЗНОЛЬ**},** // Бцент

**{**'f'**,** 4**,** **&**ЗНЕЧ**},** // Плав

**{**'d'**,** 8**,** **&**ЗНЕЧ**},** // Дво

**{**'e'**,** 12**,** **&**ЗНЕЧ**},** // Реал

**{**'o'**,** 4**,** **&**ЗНЕЧ**},** // Вплав

**{**'p'**,** 8**,** **&**ЗНЕЧ**},** // Вдво

**{**'j'**,** 12**,** **&**ЗНЕЧ**},** // Вреал

**{**'q'**,** 8**,** **&**ЗКНЕЧ**},** // Кплав

**{**'r'**,** 16**,** **&**ЗКНЕЧ**},** // Кдво

**{**'c'**,** 24**,** **&**ЗКНЕЧ**},** // Креал

**{**'v'**,** 1**},** // проц

**{**'n'**,** РНД**},** // Нет

**{**'A'**,** РА**\***2**,** **&**ЗНУЛЬ**},** // Dynamic массив

**{**'G'**,** РА**\***2**,** **&**ЗНУЛЬ**},** // Статический массив

**{**'H'**,** РА**\***2**,** **&**ЗНУЛЬ**},** // Associative массив

**{**'E'**,** РНД**},** // Перечень

**{**'S'**,** РНД**},** // Структура

**{**'C'**,** РА**,** **&**ЗНУЛЬ**},** // Класс

**{**'T'**,** РНД**},** // Типдеф

**{**'F'**,** РА**},** // Функция

**{**'D'**,** РА**\***2**,** **&**ЗНУЛЬ**},** // Делегат

**{**'P'**,** РА**,** **&**ЗНУЛЬ**},** // Указатель

**{**'R'**,** РА**,** **&**ЗНУЛЬ**},** // Ссылка

**{**'I'**,** РНД**},** // Идентификатор

**{**'?'**,** РНД**},** // Шаблон instance

**{**'B'**,** РНД**},** // Кортеж

**{**'x'**,** РНД**},** // Конст, D2

**{**'y'**,** РНД**},** // Инвариант, D2

**];**

**static** **assert(**метаИнфоТаблица**.**length **==** ТИП**.**max**+**1**);**

/// Возвращает размер of a тип.

т\_мера дайРазмер**(**Тип тип**)**

**{**

**auto** размер **=** метаИнфоТаблица**[**тип**.**тид**].**размер**;**

**if** **(**размер **==** РАЗМЕР\_НЕ\_ДОСТУПЕН**)**

**return** тип**.**sizeOf\_**();**

**return** размер**;**

**}**

**}**

/// Namespace for a установи of predefined types.

**struct** **Типы**

**{**

**static:**

/// Predefined basic types.

ТипБазовый Сим**,** Шим**,** Дим**,** Бул**,**

Байт**,** Ббайт**,** Крат**,** Бкрат**,**

Цел**,** Бцел**,** Дол**,** Бдол**,**

Цент**,** Бцент**,**

Плав**,** Дво**,** Реал**,**

Вплав**,** Вдво**,** Вреал**,**

Кплав**,** Кдво**,** Креал**,** Проц**;**

ТипБазовый Т\_мера**;** /// The размер тип.

ТипБазовый Т\_дельтаук**;** /// The pointer difference тип.

УказательТип Проц\_ук**;** /// The проц pointer тип.

ТипБазовый Ошибка**;** /// The ошибка тип.

ТипБазовый Неопределённый**;** /// The undefined тип.

ТипБазовый ПокаНеИзвестен**;** /// The символ is undefined but might be resolved.

/// Allocates an instance of ТипБазовый and assigns it в имяТипа.

**template** новТБ**(**ткст имяТипа**)**

**{**

**const** новТБ **=** **mixin(**имяТипа**~**" = new ТипБазовый(ТИП."**~**имяТипа**~**")"**);**

**}**

/// Initializes predefined types.

**static** **this()**

**{**

новТБ**!(**"Сим"**);**

новТБ**!(**"Шим"**);**

новТБ**!(**"Дим"**);**

новТБ**!(**"Бул"**);**

новТБ**!(**"Байт"**);**

новТБ**!(**"Ббайт"**);**

новТБ**!(**"Крат"**);**

новТБ**!(**"Бкрат"**);**

новТБ**!(**"Цел"**);**

новТБ**!(**"Бцел"**);**

новТБ**!(**"Дол"**);**

новТБ**!(**"Бдол"**);**

новТБ**!(**"Цент"**);**

новТБ**!(**"Бцент"**);**

новТБ**!(**"Плав"**);**

новТБ**!(**"Дво"**);**

новТБ**!(**"Реал"**);**

новТБ**!(**"Вплав"**);**

новТБ**!(**"Вдво"**);**

новТБ**!(**"Вреал"**);**

новТБ**!(**"Кплав"**);**

новТБ**!(**"Кдво"**);**

новТБ**!(**"Креал"**);**

новТБ**!(**"Проц"**);**

**version(**X86\_64**)**

**{**

Т\_мера **=** Бдол**;**

Т\_дельтаук **=** Дол**;**

**}**

**else**

**{**

Т\_мера **=** Бцел**;**

Т\_дельтаук **=** Цел**;**

**}**

Проц\_ук **=** Проц**.**укНа**;**

Ошибка **=** **new** ТипБазовый**(**ТИП**.**Ошибка**);**

Неопределённый **=** **new** ТипБазовый**(**ТИП**.**Ошибка**);**

ПокаНеИзвестен **=** **new** ТипБазовый**(**ТИП**.**Ошибка**);**

**}**

**}**

**module drc.semantic.TypesEnum;**

/// Перечень идентификаторов типов.

**enum** **ТИП**

**{**

Ошибка**,**

// Basic types.

Сим**,** /// сим

Шим**,** /// шим

Дим**,** /// дим

Бул**,** /// бул

Байт**,** /// int8

Ббайт**,** /// uint8

Крат**,** /// int16

Бкрат**,** /// uint16

Цел**,** /// int32

Бцел**,** /// uint32

Дол**,** /// int64

Бдол**,** /// uint64

Цент**,** /// int128

Бцент**,** /// uint128

Плав**,** /// float32

Дво**,** /// float64

Реал**,** /// float80

Вплав**,** /// imaginary float32

Вдво**,** /// imaginary float64

Вреал**,** /// imaginary float80

Кплав**,** /// complex float32

Кдво**,** /// complex float64

Креал**,** /// complex float80

Проц**,** /// проц

Нет**,** /// TypeNone in the specs. Why?

ДМассив**,** /// Динамический массив.

СМассив**,** /// Статический массив.

АМассив**,** /// Ассоциативный массив.

Перечень**,** /// An enum.

Структура**,** /// A struct.

Класс**,** /// A class.

Типдеф**,** /// A typedef.

Функция**,** /// A function.

Делегат**,** /// A delegate.

Указатель**,** /// A pointer.

Ссылка**,** /// A reference.

Идентификатор**,** /// An identifier.

ШЭкземпляр**,** /// Шаблон instance.

Кортеж**,** /// A template tuple.

Конст**,** /// A constant тип. D2.0

Инвариант**,** /// An invariant тип. D2.0

**}**

**Лексер (lexer)**

**module drc.lexer.Funcs;**

**import** drc**.**Unicode **:** юАльфа\_ли**;**

**const** дим \_Z\_ **=** 26**;** /// Control+Z.

/// Возвращает: да, если d является разделителем строк или абзацев Unicode.

бул симНовСтрЮ\_ли**(**дим d**)**

**{**

**return** d **==** РСд **||** d **==** РАд**;**

**}**

/// Возвращает: да, если p указывает на разделитель строки или абзаца.

бул новСтрЮ\_ли**(**сим**\*** p**)**

**{**

**return** **\***p **==** РС**[**0**]** **&&** p**[**1**]** **==** РС**[**1**]** **&&** **(**p**[**2**]** **==** РС**[**2**]** **||** p**[**2**]** **==** РА**[**2**]);**

**}**

/// Возвращает: да, если p указывает на начало новой строки Новстр.

/// $(PRE

/// Новстр := "\n" | "\r" | "\r\n" | РС | РА

/// РС := "\u2028"

/// РА := "\u2029"

/// )

бул новСтр\_ли**(**сим**\*** p**)**

**{**

**return** **\***p **==** '\n' **||** **\***p **==** '\r' **||** новСтрЮ\_ли**(**p**);**

**}**

/// Возвращает: да, если c это Новстр символ.

бул новСтр\_ли**(**дим c**)**

**{**

**return** c **==** '\n' **||** c **==** '\r' **||** симНовСтрЮ\_ли**(**c**);**

**}**

/// Возвращает: да, если p указывает на КФ символ (конец файла).

/// $(PRE

/// КФ := "\0" | \_Z\_

/// \_Z\_ := "\x1A"

/// )

бул кф\_ли**(**дим c**)**

**{**

**return** c **==** 0 **||** c **==** \_Z\_**;**

**}**

/// Возвращает: да, если p указывает на первый символ EndOfLine.

/// $(PRE EndOfLine := Новстр | КФ)

бул конецСтроки\_ли**(**сим**\*** p**)**

**{**

**return** новСтр\_ли**(**p**)** **||** кф\_ли**(\***p**);**

**}**

/// Сканирует символ Новстр и устанавливает p на символ после него.

/// Возвращает: да, если он найден или нет в прот.сл.

бул сканируйНовСтр**(**ref сим**\*** p**)**

**in** **{** **assert(**p**);** **}**

**body**

**{**

**switch** **(\***p**)**

**{**

**case** '\r'**:**

**if** **(**p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**++**p**;**

**break;**

**default:**

**if** **(**новСтрЮ\_ли**(**p**))**

p **+=** 3**;**

**else**

**return** нет**;**

**}**

**return** да**;**

**}**

/// Сканирует символ Новстр и устанавливает p на символ после него.

/// Возвращает: да, если он найден или нет в прот.сл.

бул сканируйНовСтр**(**ref сим**\*** p**,** сим**\*** конец**)**

**in** **{** **assert(**p **&&** p **<** конец**);** **}**

**body**

**{**

**switch** **(\***p**)**

**{**

**case** '\r'**:**

**if** **(**p**+**1 **<** конец **&&** p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**++**p**;**

**break;**

**default:**

**if** **(**p**+**2 **<** конец **&&** новСтрЮ\_ли**(**p**))**

p **+=** 3**;**

**else**

**return** нет**;**

**}**

**return** да**;**

**}**

/// Сканирует Новстр в обрантом направлении и устанавливает конец

/// на первый символ нс.

/// Возвращает: да, если он найден или нет в прот.сл.

бул сканируйНовСтрРеверс**(**сим**\*** начало**,** ref сим**\*** конец**)**

**{**

**switch** **(\***конец**)**

**{**

**case** '\n'**:**

**if** **(**начало **<=** конец**-**1 **&&** конец**[-**1**]** **==** '\r'**)**

конец**--;**

**case** '\r'**:**

**break;**

**case** РС**[**2**],** РА**[**2**]:**

**if** **(**начало **<=** конец**-**2 **&&** конец**[-**1**]** **==** РС**[**1**]** **&&** конец**[-**2**]** **==** РС**[**0**])** **{**

конец **-=** 2**;**

**break;**

**}**

// fall through

**default:**

**return** нет**;**

**}**

**return** да**;**

**}**

/// Сканирует идентификатор D.

/// Параметры:

/// ref\_p = откуда начать.

/// конец = где закончить.

/// Возвращает: идентификатор, если он действителен (устанавливает ref\_p на 1 после ид,) или

/// null, если недействителен (оставляет неизменным ref\_p.)

ткст сканируйИдентификатор**(**ref сим**\*** ref\_p**,** сим**\*** конец**)**

**in** **{** **assert(**ref\_p **&&** ref\_p **<** конец**);** **}**

**body**

**{**

**auto** p **=** ref\_p**;**

**if** **(**начсим\_ли**(\***p**)** **||** юАльфа\_ли**(**p**,** конец**))** // IdStart

**{**

**do** // IdChar\*

p**++;**

**while** **(**p **<** конец **&&** **(**идент\_ли**(\***p**)** **||** юАльфа\_ли**(**p**,** конец**)))**

**auto** identifier **=** ref\_p**[**0 **..** p**-**ref\_p**];**

ref\_p **=** p**;**

**return** identifier**;**

**}**

**return** **null;**

**}**

/// Таблица свойств символов ASCII.

**static** **const** цел ptable**[**256**]** **=** **[**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,**32**,** 0**,**32**,**32**,** 0**,** 0**,** 0**,**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,**

32**,** 0**,** 0x2200**,** 0**,** 0**,** 0**,** 0**,** 0x2700**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,**

7**,** 7**,** 7**,** 7**,** 7**,** 7**,** 7**,** 7**,** 6**,** 6**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0x3f00**,**

0**,**12**,**12**,**12**,**12**,**12**,**12**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,**

8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 0**,** 0x5c00**,** 0**,** 0**,**16**,**

0**,** 0x70c**,** 0x80c**,**12**,**12**,**12**,** 0xc0c**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 8**,** 0xa08**,** 8**,**

8**,** 8**,** 0xd08**,** 8**,** 0x908**,** 8**,** 0xb08**,** 8**,** 8**,** 8**,** 8**,** 0**,** 0**,** 0**,** 0**,** 0**,**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,**

0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0**,** 0

**];**

/// Перечень флагов свойств символов.

**enum** **СвойствоС**

**{**

Восмиричный **=** 1**,** /// 0-7

Десятичный **=** 1**<<**1**,** /// 0-9

Гекс **=** 1**<<**2**,** /// 0-9a-fA-F

Буква **=** 1**<<**3**,** /// a-zA-Z

Подчерк **=** 1**<<**4**,** /// \_

Пробельный **=** 1**<<**5 /// ' ' \t \v \f

**}**

**const** бцел EVMask **=** 0xFF00**;** // Bit mask for escape значение.

**private** **alias** СвойствоС CP**;**

/// Возвращает: да if c is an octal digit.

цел восмир\_ли**(**сим c**)** **{** **return** ptable**[**c**]** **&** CP**.**Восмиричный**;** **}**

/// Возвращает: да if c is a decimal digit.

цел цифра\_ли**(**сим c**)** **{** **return** ptable**[**c**]** **&** CP**.**Десятичный**;** **}**

/// Возвращает: да if c is a hexadecimal digit.

цел гекс\_ли**(**сим c**)** **{** **return** ptable**[**c**]** **&** CP**.**Гекс**;** **}**

/// Возвращает: да if c is a letter.

цел буква\_ли**(**сим c**)** **{** **return** ptable**[**c**]** **&** CP**.**Буква**;** **}**

/// Возвращает: да if c is an alphanumeric.

цел цифробукв\_ли**(**сим c**)** **{** **return** ptable**[**c**]** **&** **(**CP**.**Буква **|** CP**.**Десятичный**);** **}**

/// Возвращает: да if c is the beginning of a D identifier (only ASCII.)

цел начсим\_ли**(**сим c**)** **{** **return** ptable**[**c**]** **&** **(**CP**.**Буква **|** CP**.**Подчерк**);** **}**

/// Возвращает: да if c is a D identifier символ (only ASCII.)

цел идент\_ли**(**сим c**)** **{** **return** ptable**[**c**]** **&** **(**CP**.**Буква **|** CP**.**Подчерк **|** CP**.**Десятичный**);** **}**

/// Возвращает: да if c is a whitespace символ.

цел пбел\_ли**(**сим c**)** **{** **return** ptable**[**c**]** **&** CP**.**Пробельный**;** **}**

/// Возвращает: the escape значение for c.

цел сим8еск**(**сим c**)** **{** **return** ptable**[**c**]** **>>** 8**;** /\*(ptable[c] & EVMask) >> 8;\*/ **}**

/// Возвращает: да if c is an ASCII символ.

цел аски\_ли**(**бцел c**)** **{** **return** c **<** 128**;** **}**

**version(**gen\_ptable**)**

**static** **this()**

**{**

**alias** ptable p**;**

**assert(**p**.**length **==** 256**);**

// Initialize символ properties таблица.

**for** **(**цел i**;** i **<** p**.**length**;** **++**i**)**

**{**

p**[**i**]** **=** 0**;** // Reset

**if** **(**'0' **<=** i **&&** i **<=** '7'**)**

p**[**i**]** **|=** CP**.**Восмиричный**;**

**if** **(**'0' **<=** i **&&** i **<=** '9'**)**

p**[**i**]** **|=** CP**.**Десятичный **|** CP**.**Гекс**;**

**if** **(**'a' **<=** i **&&** i **<=** 'f' **||** 'A' **<=** i **&&** i **<=** 'F'**)**

p**[**i**]** **|=** CP**.**Гекс**;**

**if** **(**'a' **<=** i **&&** i **<=** 'z' **||** 'A' **<=** i **&&** i **<=** 'Z'**)**

p**[**i**]** **|=** CP**.**Буква**;**

**if** **(**i **==** '\_'**)**

p**[**i**]** **|=** CP**.**Подчерк**;**

**if** **(**i **==** ' ' **||** i **==** '\t' **||** i **==** '\v' **||** i **==** '\f'**)**

p**[**i**]** **|=** CP**.**Пробельный**;**

**}**

// Store escape sequence значения in second байт.

**assert(**СвойствоС**.**max **<=** ббайт**.**max**,** "символ property флаги and escape значение байт overlap."**);**

p**[**'\''**]** **|=** 39 **<<** 8**;**

p**[**'"'**]** **|=** 34 **<<** 8**;**

p**[**'?'**]** **|=** 63 **<<** 8**;**

p**[**'\\'**]** **|=** 92 **<<** 8**;**

p**[**'a'**]** **|=** 7 **<<** 8**;**

p**[**'b'**]** **|=** 8 **<<** 8**;**

p**[**'f'**]** **|=** 12 **<<** 8**;**

p**[**'n'**]** **|=** 10 **<<** 8**;**

p**[**'r'**]** **|=** 13 **<<** 8**;**

p**[**'t'**]** **|=** 9 **<<** 8**;**

p**[**'v'**]** **|=** 11 **<<** 8**;**

// Print a formatted массив literal.

ткст массив **=** "[\n"**;**

**foreach** **(**i**,** c**;** ptable**)**

**{**

массив **~=** Формат**((**c**>**255**?**" 0x{0:x},"**:**"{0,2},"**),** c**)** **~** **(((**i**+**1**)** **%** 16**)** **?** ""**:**"\n"**);**

**}**

массив**[**$**-**2**..**$**]** **=** "\n]"**;**

выдай**(**массив**).**нс**;**

**}**

**module drc.lexer.Identifier;**

**import** drc**.**lexer**.**TokensEnum**,**

drc**.**lexer**.**IdentsEnum**;**

**import** common**;**

/// Представляет идентификатор по определению в спецификации D.

///

/// $(PRE

/// Идентификатор := НачалоИд СимвИд\*

/// НачалоИд := "\_" | Буква

/// СимвИд := НачалоИд | "0"-"9"

/// Буква := ЮАльфа

/// )

/// See\_Also:

/// Алфавитные символы Unicode определены в Unicode 5.0.0.

**align(**1**)**

**struct** **Идентификатор**

**{**

ткст ткт**;** /// Идентификатор в ткст UTF-8.

TOK вид**;** /// Вид семы.

ВИД видИд**;** /// Только для предопределённых идентификаторов.

**static** Идентификатор**\*** opCall**(**ткст ткт**,** TOK вид**)**

**{**

**auto** ид **=** **new** Идентификатор**;**

ид**.**ткт **=** ткт**;**

ид**.**вид **=** вид**;**

**return** ид**;**

**}**

**static** Идентификатор**\*** opCall**(**ткст ткт**,** TOK вид**,** ВИД видИд**)**

**{**

**auto** ид **=** **new** Идентификатор**;**

ид**.**ткт **=** ткт**;**

ид**.**вид **=** вид**;**

ид**.**видИд **=** видИд**;**

**return** ид**;**

**}**

бцел вХэш**()**

**{**

бцел хэш**;**

**foreach(**c**;** ткт**)** **{**

хэш **\*=** 11**;**

хэш **+=** c**;**

**}**

**return** хэш**;**

**}**

**}**

// pragma(сооб, Идентификатор.sizeof.stringof);

**module drc.lexer.IdentsEnum;**

**import** drc**.**lexer**.**IdentsGenerator**;**

**version(**DDoc**)**

**enum** ВИД **:** бкрат**;** /// Перечень видов предопределенных идентификаторов.

**else**

**mixin(**

// Перечисляет предопределённые идентификаторы.

"enum ВИД : бкрат {"

"Нуль,"

**~** генерируйЧленыИД **~**

"}"

**);**

**module drc.lexer.IdentsGenerator;**

/// Таблица предопределенных идентификаторов.

///

/// Формат ('#' начинает комментарии):

/// $(PRE

/// ПредопределенныйИдентификатор := ИмяИсхКода (":" ТекстИда)?

/// ИмяИсхКода := Идентификатор # Имя, которое будет использоваться в исходном коде.

/// ТекстИда := Пусто | Идентификатор # Действительный текст идентификатора.

/// Пусто := "" # ТекстИда может быть пустым.

/// Идентификатор := см. модуль $(MODLINK drc.lexer.Identifier).

/// )

/// Если ТекстИда не указан, то дефолтом является ИмяИсхКода.

**private** **static** **const** сим**[][]** предопрИденты **=** **[**

// Специальный пустой идентификатор:

"Пусто:"**,**

// Предопределенные идентификаторы версии:

"DigitalMars"**,** "X86"**,** "X86\_64"**,**

/\*"Windows", \*/"Win32"**,** "Win64"**,**

"Linux:linux"**,** "ЛитлЭндиан"**,** "BigEndian"**,**

"D\_Coverage"**,** "D\_InlineAsm\_X86"**,** "D\_Version2"**,**

"none"**,** "all"**,**

// Вариадические параметры:

"Аргументы:\_arguments"**,** "Аргук:\_argptr"**,**

// масштаб(Идентификатор):

"выход"**,** "успех"**,** "сбой"**,** "exit"**,** "success"**,** "failure"**,**

// Прагмы:

"сооб"**,** "lib"**,** "startaddress"**,** "msg"**,**

// Конпоновка:

"C"**,** "D"**,** "Windows"**,** "Pascal"**,** "System"**,**

// Con-/Destructor:

"Ктор:\_\_ctor"**,** "Дтор:\_\_dtor"**,**

// new() and delete() methods.

"Нов:\_\_new"**,** "Удалить:\_\_delete"**,**

// Юниттест and invariant.

"Юниттест:\_\_unittest"**,** "Инвариант:\_\_invariant"**,**

// Методы перегрузки операторов

"opNeg"**,** "opPos"**,** "opCom"**,**

"opEquals"**,** "opCmp"**,** "opAssign"**,**

"opAdd"**,** "opAdd\_r"**,** "opAddAssign"**,**

"opSub"**,** "opSub\_r"**,** "opSubAssign"**,**

"opMul"**,** "opMul\_r"**,** "opMulAssign"**,**

"opDiv"**,** "opDiv\_r"**,** "opDivAssign"**,**

"opMod"**,** "opMod\_r"**,** "opModAssign"**,**

"opAnd"**,** "opAnd\_r"**,** "opAndAssign"**,**

"opOr"**,** "opOr\_r"**,** "opOrAssign"**,**

"opXor"**,** "opXor\_r"**,** "opXorAssign"**,**

"opShl"**,** "opShl\_r"**,** "opShlAssign"**,**

"opShr"**,** "opShr\_r"**,** "opShrAssign"**,**

"opUShr"**,** "opUShr\_r"**,** "opUShrAssign"**,**

"opCat"**,** "opCat\_r"**,** "opCatAssign"**,**

"opIn"**,** "opIn\_r"**,**

"opIndex"**,** "opIndexAssign"**,**

"opSlice"**,** "opSliceAssign"**,**

"opPostInc"**,**

"opPostDec"**,**

"opCall"**,**

"opCast"**,**

"opStar"**,** // D2

// foreach and foreach\_reverse:

"opApply"**,** "opApplyReverse"**,**

// Entry function:

"main"**,**

// ASM identifiers:

"near"**,** "far"**,** "word"**,** "dword"**,** "qword"**,**

"ptr"**,** "offset"**,** "seg"**,** "\_\_LOCAL\_SIZE"**,**

"FS"**,** "ST"**,**

"AL"**,** "AH"**,** "AX"**,** "EAX"**,**

"BL"**,** "BH"**,** "BX"**,** "EBX"**,**

"CL"**,** "CH"**,** "CX"**,** "ECX"**,**

"DL"**,** "DH"**,** "DX"**,** "EDX"**,**

"BP"**,** "EBP"**,** "SP"**,** "ESP"**,**

"DI"**,** "EDI"**,** "SI"**,** "ESI"**,**

"ES"**,** "CS"**,** "SS"**,** "DS"**,** "GS"**,**

"CR0"**,** "CR2"**,** "CR3"**,** "CR4"**,**

"DR0"**,** "DR1"**,** "DR2"**,** "DR3"**,** "DR6"**,** "DR7"**,**

"TR3"**,** "TR4"**,** "TR5"**,** "TR6"**,** "TR7"**,**

"MM0"**,** "MM1"**,** "MM2"**,** "MM3"**,**

"MM4"**,** "MM5"**,** "MM6"**,** "MM7"**,**

"XMM0"**,** "XMM1"**,** "XMM2"**,** "XMM3"**,**

"XMM4"**,** "XMM5"**,** "XMM6"**,** "XMM7"**,**

**];**

сим**[][]** дайПару**(**ткст текстИда**)**

**{**

**foreach** **(**i**,** c**;** текстИда**)**

**if** **(**c **==** ':'**)**

**return** **[**текстИда**[**0**..**i**],** текстИда**[**i**+**1**..**текстИда**.**length**]];**

**return** **[**текстИда**,** текстИда**];**

**}**

**unittest**

**{**

**static** **assert(**

дайПару**(**"тест"**)** **==** **[**"тест"**,** "тест"**]** **&&**

дайПару**(**"тест:tset"**)** **==** **[**"тест"**,** "tset"**]** **&&**

дайПару**(**"empty:"**)** **==** **[**"empty"**,** ""**]**

**);**

**}**

/++

CTF для генерации членов структуры Идент.

Результирующий текст выглядить примерно так:

---

private struct Иды {static const:

Идентификатор \_Empty = {"", TOK.Идентификатор, ВИД.Пусто};

Идентификатор \_main = {"main", TOK.Идентификатор, ВИД.main};

// etc.

}

Идентификатор\* Пусто = &Иды.\_Empty;

Идентификатор\* main = &Иды.\_main;

// etc.

private Идентификатор\*[] \_\_allIds = [

Пусто,

main,

// и т.д.

];

---

+/

ткст генерируйЧленыИдент**()**

**{**

ткст приват\_члены **=** "private struct Иды {static const:"**;**

ткст публ\_члены **=** ""**;**

ткст массив **=** "private Идентификатор\*[] \_\_allIds = ["**;**

**foreach** **(**идент**;** предопрИденты**)**

**{**

сим**[][]** пара **=** дайПару**(**идент**);**

// Идентификатор \_name = {"имя", TOK.Идентификатор, ID.имя};

приват\_члены **~=** "Идентификатор \_"**~**пара**[**0**]~**` = {"`**~**пара**[**1**]~**`", TOK.Идентификатор, ВИД.`**~**пара**[**0**]~**"};\n"**;**

// Идентификатор\* имя = &\_name;

публ\_члены **~=** "Идентификатор\* "**~**пара**[**0**]~**" = &Иды.\_"**~**пара**[**0**]~**";\n"**;**

массив **~=** пара**[**0**]~**","**;**

**}**

приват\_члены **~=** "}"**;** // Close private {

массив **~=** "];"**;**

**return** приват\_члены **~** публ\_члены **~** массив**;**

**}**

/// CTF for generating the члены of the enum ВИД.

ткст генерируйЧленыИД**()**

**{**

ткст члены**;**

**foreach** **(**идент**;** предопрИденты**)**

члены **~=** дайПару**(**идент**)[**0**]** **~** ",\n"**;**

**return** члены**;**

**}**

// pragma(сооб, генерируйЧленыИдент());

// pragma(сооб, генерируйЧленыИД());

**module drc.lexer.IdTable;**

**import** drc**.**lexer**.**TokensEnum**,**

drc**.**lexer**.**IdentsGenerator**,**

drc**.**lexer**.**Keywords**;**

**import** common**;**

**public** **import** drc**.**lexer**.**Identifier**,**

drc**.**lexer**.**IdentsEnum**;**

/// Неймспейс для предопределенных идентификаторов.

**struct** **Идент**

**{**

**const** **static**

**{**

**mixin(**генерируйЧленыИдент**());**

**}**

/// Возвращает массив предопределенных идентификаторов.

**static** Идентификатор**\*[]** всеИды**()**

**{**

**return** \_\_allIds**;**

**}**

**}**

/// Глобальная таблица для размещения и получения идентификаторов.

**struct** **ТаблицаИд**

**{**

**static:**

/// Набор общих, предопределенных идентификаторов для быстрых поисков.

**private** Идентификатор**\*[**ткст**]** статическаяТаблица**;**

/// Таблица, растущая с каждым новым уникальным идентификатором.

**private** Идентификатор**\*[**ткст**]** растущаяТаблица**;**

/// Загружает ключевые слова и предопределенные идентификаторы в статическую таблицу.

**static** **this()**

**{**

**foreach** **(**ref k**;** g\_reservedIds**)**

статическаяТаблица**[**k**.**ткт**]** **=** **&**k**;**

**foreach** **(**ид**;** Идент**.**всеИды**())**

статическаяТаблица**[**ид**.**ткт**]** **=** ид**;**

статическаяТаблица**.**rehash**;**

**}**

/// Ищет ткстИда в обеих таблицах.

Идентификатор**\*** сыщи**(**ткст ткстИда**)**

**{**

**auto** ид **=** вСтатической**(**ткстИда**);**

**if** **(**ид**)**

**return** ид**;**

**return** вРастущей**(**ткстИда**);**

**}**

/// Ищет ткстИда в статической таблице.

Идентификатор**\*** вСтатической**(**ткст ткстИда**)**

**{**

**auto** ид **=** ткстИда **in** статическаяТаблица**;**

**return** ид **?** **\***ид **:** **null;**

**}**

**alias** Идентификатор**\*** **function(**ткст ткстИда**)** ФункцияПоиска**;**

/// Ищет ткстИда в растущей таблице.

ФункцияПоиска вРастущей **=** **&**\_inGrowing\_unsafe**;** // Дефолт на небезопасную функцию.

/// Устанавливает режим безопасности нити для растущей таблицы.

проц установиНитебезопасность**(**бул b**)**

**{**

**if** **(**b**)**

вРастущей **=** **&**\_inGrowing\_safe**;**

**else**

вРастущей **=** **&**\_inGrowing\_unsafe**;**

**}**

/// Возвращает да, если доступ к растущей таблице нитебезопасен.

бул нитебезопасно\_ли**()**

**{**

**return** вРастущей **is** **&**\_inGrowing\_safe**;**

**}**

/// Ищет ткстИда в таблице.

///

/// Добавляет ткстИда в таблицу, если он не найден.

**private** Идентификатор**\*** \_inGrowing\_unsafe**(**ткст ткстИда**)**

**out(**ид**)**

**{** **assert(**ид **!is** **null);** **}**

**body**

**{**

**auto** ид **=** ткстИда **in** растущаяТаблица**;**

**if** **(**ид**)**

**return** **\***ид**;**

**auto** newID **=** Идентификатор**(**ткстИда**,** TOK**.**Идентификатор**);**

растущаяТаблица**[**ткстИда**]** **=** newID**;**

**return** newID**;**

**}**

/// Ищет ткстИда в таблица.

///

/// Добавляет ткстИда в таблицу, если не найден.

/// Доступ в структуру данных синхронизирован.

**private** Идентификатор**\*** \_inGrowing\_safe**(**ткст ткстИда**)**

**{**

**synchronized**

**return** \_inGrowing\_unsafe**(**ткстИда**);**

**}**

/+

Идентификатор\* addIdentifiers(сим[][] idStrings)

{

auto ids = new Идентификатор\*[idStrings.length];

foreach (i, ткстИда; idStrings)

{

Идентификатор\*\* ид = ткстИда in tabulatedIds;

if (!ид)

{

auto newID = Идентификатор(TOK.Идентификатор, ткстИда);

tabulatedIds[ткстИда] = newID;

ид = &newID;

}

ids[i] = \*ид;

}

}

+/

**static** бцел anonCount**;** /// Счётчик бля безымянных идентификаторов.

/// Генерирует безымянный идентификатор.

///

/// Конкатенирует префикс с anonCount.

/// Этот идентификатор не помещён в таблицу.

Идентификатор**\*** генБезымянныйИД **(**ткст prefix**)**

**{**

**++**anonCount**;**

**auto** x **=** anonCount**;**

// Convert счёт в a ткст and добавь it в ткт.

ткст чис**;**

**do**

чис **=** **cast(**сим**)(**'0' **+** **(**x **%** 10**))** **~** чис**;**

**while** **(**x **/=** 10**)**

**return** Идентификатор**(**prefix **~** чис**,** TOK**.**Идентификатор**);**

**}**

/// Генерирует идентификатор для безымянного перечня.

Идентификатор**\*** генИДАнонПеречня**()**

**{**

**return** генБезымянныйИД **(**"\_\_anonenum"**);**

**}**

/// Генерирует идентификатор для безымянного класса.

Идентификатор**\*** genAnonClassID**()**

**{**

**return** генБезымянныйИД **(**"\_\_anonclass"**);**

**}**

/// Генерирует идентификатор для безымянной структуры.

Идентификатор**\*** genAnonStructID**()**

**{**

**return** генБезымянныйИД **(**"\_\_anonstruct"**);**

**}**

/// Генерирует идентификатор для анонимного союза.

Идентификатор**\*** genAnonUnionID**()**

**{**

**return** генБезымянныйИД **(**"\_\_anonunion"**);**

**}**

/// Генерирует идентификатор для модуля, у которого не указано имя.

Идентификатор**\*** генИдМодуля**()**

**{**

**return** генБезымянныйИД **(**"\_\_module"**);**

**}**

**}**

**unittest**

**{**

// TODO: пиши benchmark.

// Single таблица

// Single таблица. synchronized

// Two tables.

// Two tables. synchronized

**}**

**module drc.lexer.Keywords;**

**import** drc**.**lexer**.**Token**,**

drc**.**lexer**.**Identifier**;**

/// Таблица резервированных идентификаторов.

**static** **const** Идентификатор**[]** g\_reservedIds **=** **[**

**{**"abstract"**,** TOK**.**Абстрактный**},**

**{**"абстрактный"**,** TOK**.**Абстрактный**},**

**{**"alias"**,** TOK**.**Алиас**},**

**{**"иной"**,** TOK**.**Алиас**},**

**{**"align"**,** TOK**.**Расклад**},**

**{**"расклад"**,** TOK**.**Расклад**},**

**{**"asm"**,** TOK**.**Асм**},**

**{**"асм"**,** TOK**.**Асм**},**

**{**"assert"**,** TOK**.**Подтвердить**},**

**{**"подтверди"**,** TOK**.**Подтвердить**},**

**{**"auto"**,** TOK**.**Авто**},**

**{**"авто"**,** TOK**.**Авто**},**

**{**"body"**,** TOK**.**Тело**},**

**{**"тело"**,** TOK**.**Тело**},**

**{**"bool"**,** TOK**.**Бул**},**

**{**"бул"**,** TOK**.**Бул**},**

**{**"break"**,** TOK**.**Всё**},**

**{**"всё"**,** TOK**.**Всё**},**

**{**"byte"**,** TOK**.**Байт**},**

**{**"байт"**,** TOK**.**Байт**},**

**{**"case"**,** TOK**.**Реле**},**

**{**"реле"**,** TOK**.**Реле**},**

**{**"cast"**,** TOK**.**Каст**},**

**{**"catch"**,** TOK**.**Кэтч**},**

**{**"cdouble"**,** TOK**.**Кдво**},**

**{**"кдво"**,** TOK**.**Кдво**},**

**{**"cent"**,** TOK**.**Цент**},**

**{**"цент"**,** TOK**.**Цент**},**

**{**"cfloat"**,** TOK**.**Кплав**},**

**{**"кплав"**,** TOK**.**Кплав**},**

**{**"char"**,** TOK**.**Сим**},**

**{**"сим"**,** TOK**.**Сим**},**

**{**"class"**,** TOK**.**Класс**},**

**{**"класс"**,** TOK**.**Класс**},**

**{**"const"**,** TOK**.**Конст**},**

**{**"конст"**,** TOK**.**Конст**},**

**{**"continue"**,** TOK**.**Далее**},**

**{**"далее"**,** TOK**.**Далее**},**

**{**"creal"**,** TOK**.**Креал**},**

**{**"креал"**,** TOK**.**Креал**},**

**{**"dchar"**,** TOK**.**Дим**},**

**{**"дим"**,** TOK**.**Дим**},**

**{**"debug"**,** TOK**.**Отладка**},**

**{**"отладка"**,** TOK**.**Отладка**},**

**{**"default"**,** TOK**.**Дефолт**},**

**{**"дефолт"**,** TOK**.**Дефолт**},**

**{**"delegate"**,** TOK**.**Делегат**},**

**{**"делегат"**,** TOK**.**Делегат**},**

**{**"delete"**,** TOK**.**Удалить**},**

**{**"удали"**,** TOK**.**Удалить**},**

**{**"deprecated"**,** TOK**.**Устаревший**},**

**{**"устаревший"**,** TOK**.**Устаревший**},**

**{**"do"**,** TOK**.**Делай**},**

**{**"делай"**,** TOK**.**Делай**},**

**{**"double"**,** TOK**.**Дво**},**

**{**"дво"**,** TOK**.**Дво**},**

**{**"else"**,** TOK**.**Иначе**},**

**{**"иначе"**,** TOK**.**Иначе**},**

**{**"enum"**,** TOK**.**Перечень**},**

**{**"перечень"**,** TOK**.**Перечень**},**

**{**"export"**,** TOK**.**Экспорт**},**

**{**"экспорт"**,** TOK**.**Экспорт**},**

**{**"extern"**,** TOK**.**Экстерн**},**

**{**"экстерн"**,** TOK**.**Экстерн**},**

**{**"false"**,** TOK**.**Ложь**},**

**{**"нет"**,** TOK**.**Ложь**},**

**{**"final"**,** TOK**.**Окончательный**},**

**{**"окончательный"**,** TOK**.**Окончательный**},**

**{**"finally"**,** TOK**.**Finally**},**

**{**"наконец"**,** TOK**.**Finally**},**

**{**"float"**,** TOK**.**Плав**},**

**{**"плав"**,** TOK**.**Плав**},**

**{**"for"**,** TOK**.**При**},**

**{**"при"**,** TOK**.**При**},**

**{**"foreach"**,** TOK**.**Длявсех**},**

**{**"длявсех"**,** TOK**.**Длявсех**},**

**{**"foreach\_reverse"**,** TOK**.**Длявсех\_реверс**},**

**{**"длявсехрев"**,** TOK**.**Длявсех\_реверс**},**

**{**"function"**,** TOK**.**Функция**},**

**{**"функ"**,** TOK**.**Функция**},**

**{**"goto"**,** TOK**.**Переход**},**

**{**"переход\_на"**,** TOK**.**Переход**},**

**{**"idouble"**,** TOK**.**Вдво**},**

**{**"вдво"**,** TOK**.**Вдво**},**

**{**"if"**,** TOK**.**Если**},**

**{**"если"**,** TOK**.**Если**},**

**{**"ifloat"**,** TOK**.**Вплав**},**

**{**"вплав"**,** TOK**.**Вплав**},**

**{**"import"**,** TOK**.**Импорт**},**

**{**"импорт"**,** TOK**.**Импорт**},**

**{**"in"**,** TOK**.**Вхо**},**

**{**"вхо"**,** TOK**.**Вхо**},**

**{**"inout"**,** TOK**.**Вховых**},**

**{**"вховых"**,** TOK**.**Вховых**},**

**{**"int"**,** TOK**.**Цел**},**

**{**"цел"**,** TOK**.**Цел**},**

**{**"interface"**,** TOK**.**Интерфейс**},**

**{**"интерфейс"**,** TOK**.**Интерфейс**},**

**{**"invariant"**,** TOK**.**Инвариант**},**

**{**"инвариант"**,** TOK**.**Инвариант**},**

**{**"ireal"**,** TOK**.**Вреал**},**

**{**"вреал"**,** TOK**.**Вреал**},**

**{**"is"**,** TOK**.**Является**},**

**{**"есть"**,** TOK**.**Является**},**

**{**"lazy"**,** TOK**.**Отложенный**},**

**{**"отложеный"**,** TOK**.**Отложенный**},**

**{**"long"**,** TOK**.**Дол**},**

**{**"дол"**,** TOK**.**Дол**},**

**{**"macro"**,** TOK**.**Макрос**},** // D2.0

**{**"макро"**,** TOK**.**Макрос**},**

**{**"mixin"**,** TOK**.**Смесь**},**

**{**"впиши"**,** TOK**.**Смесь**},**

**{**"module"**,** TOK**.**Модуль**},**

**{**"модуль"**,** TOK**.**Модуль**},**

**{**"new"**,** TOK**.**Нов**},**

**{**"нов"**,** TOK**.**Нов**},**

**{**"nothrow"**,** TOK**.**Nothrow**},** // D2.0

**{**"null"**,** TOK**.**Нуль**},**

**{**"пусто"**,** TOK**.**Нуль**},**

**{**"out"**,** TOK**.**Вых**},**

**{**"вых"**,** TOK**.**Вых**},**

**{**"override"**,** TOK**.**Перепись**},**

**{**"перепись"**,** TOK**.**Перепись**},**

**{**"package"**,** TOK**.**Пакет**},**

**{**"пакет"**,** TOK**.**Пакет**},**

**{**"pragma"**,** TOK**.**Прагма**},**

**{**"прагма"**,** TOK**.**Прагма**},**

**{**"private"**,** TOK**.**Приватный**},**

**{**"protected"**,** TOK**.**Защищённый**},**

**{**"public"**,** TOK**.**Публичный**},**

**{**"pure"**,** TOK**.**Pure**},** // D2.0

**{**"real"**,** TOK**.**Реал**},**

**{**"реал"**,** TOK**.**Реал**},**

**{**"ref"**,** TOK**.**Реф**},**

**{**"return"**,** TOK**.**Итог**},**

**{**"итог"**,** TOK**.**Итог**},**

**{**"scope"**,** TOK**.**Масштаб**},**

**{**"short"**,** TOK**.**Крат**},**

**{**"крат"**,** TOK**.**Крат**},**

**{**"static"**,** TOK**.**Статический**},**

**{**"struct"**,** TOK**.**Структура**},**

**{**"структ"**,** TOK**.**Структура**},**

**{**"super"**,** TOK**.**Супер**},**

**{**"супер"**,** TOK**.**Супер**},**

**{**"switch"**,** TOK**.**Щит**},**

**{**"щит"**,** TOK**.**Щит**},**

**{**"synchronized"**,** TOK**.**Синхронизованный**},**

**{**"синхронно"**,** TOK**.**Синхронизованный**},**

**{**"template"**,** TOK**.**Шаблон**},**

**{**"шаблон"**,** TOK**.**Шаблон**},**

**{**"this"**,** TOK**.**Этот**},**

**{**"этот"**,** TOK**.**Этот**},**

**{**"throw"**,** TOK**.**Брось**},**

**{**"брось"**,** TOK**.**Брось**},**

**{**"\_\_traits"**,** TOK**.**Трэтс**},** // D2.0

**{**"true"**,** TOK**.**Истина**},**

**{**"да"**,** TOK**.**Истина**},**

**{**"try"**,** TOK**.**Пробуй**},**

**{**"пробуй"**,** TOK**.**Пробуй**},**

**{**"typedef"**,** TOK**.**Типдеф**},**

**{**"typeid"**,** TOK**.**Идтипа**},**

**{**"typeof"**,** TOK**.**Типа**},**

**{**"ubyte"**,** TOK**.**Ббайт**},**

**{**"ббайт"**,** TOK**.**Ббайт**},**

**{**"ucent"**,** TOK**.**Бцент**},**

**{**"бцент"**,** TOK**.**Бцент**},**

**{**"uint"**,** TOK**.**Бцел**},**

**{**"бцел"**,** TOK**.**Бцел**},**

**{**"ulong"**,** TOK**.**Бдол**},**

**{**"бдол"**,** TOK**.**Бдол**},**

**{**"union"**,** TOK**.**Союз**},**

**{**"союз"**,** TOK**.**Союз**},**

**{**"unittest"**,** TOK**.**Юниттест**},**

**{**"ushort"**,** TOK**.**Бкрат**},**

**{**"бкрат"**,** TOK**.**Бкрат**},**

**{**"version"**,** TOK**.**Версия**},**

**{**"версия"**,** TOK**.**Версия**},**

**{**"void"**,** TOK**.**Проц**},**

**{**"проц"**,** TOK**.**Проц**},**

**{**"volatile"**,** TOK**.**Волатайл**},**

**{**"wchar"**,** TOK**.**Шим**},**

**{**"шим"**,** TOK**.**Шим**},**

**{**"while"**,** TOK**.**Пока**},**

**{**"пока"**,** TOK**.**Пока**},**

**{**"with"**,** TOK**.**Для**},**

**{**"для"**,** TOK**.**Для**},**

// Special семы:

**{**"\_\_FILE\_\_"**,** TOK**.**ФАЙЛ**},**

**{**"\_\_ФАЙЛ\_\_"**,** TOK**.**ФАЙЛ**},**

**{**"\_\_LINE\_\_"**,** TOK**.**СТРОКА**},**

**{**"\_\_СТРОКА\_\_"**,** TOK**.**СТРОКА**},**

**{**"\_\_DATE\_\_"**,** TOK**.**ДАТА**},**

**{**"\_\_ДАТА\_\_"**,** TOK**.**ДАТА**},**

**{**"\_\_TIME\_\_"**,** TOK**.**ВРЕМЯ**},**

**{**"\_\_ВРЕМЯ\_\_"**,** TOK**.**ВРЕМЯ**},**

**{**"\_\_TIMESTAMP\_\_"**,** TOK**.**ШТАМПВРЕМЕНИ**},**

**{**"\_\_ШТАМПВРЕМЕНИ\_\_"**,** TOK**.**ШТАМПВРЕМЕНИ**},**

**{**"\_\_VENDOR\_\_"**,** TOK**.**ПОСТАВЩИК**},**

**{**"\_\_ПОСТАВЩИК\_\_"**,** TOK**.**ПОСТАВЩИК**},**

**{**"\_\_VERSION\_\_"**,** TOK**.**ВЕРСИЯ**},**

**{**"\_\_ВЕРСИЯ\_\_"**,** TOK**.**ВЕРСИЯ**},**

**{**"\_\_EOF\_\_"**,** TOK**.**КФ**},** // D2.0

**{**"\_\_КФ\_\_"**,** TOK**.**КФ**},**

**];**

**module drc.lexer.Lexer;**

**import** drc**.**lexer**.**Token**,**

drc**.**lexer**.**Keywords**,**

drc**.**lexer**.**Identifier**,**

drc**.**lexer**.**IdTable**;**

**import** drc**.**Diagnostics**;**

**import** drc**.**Messages**;**

**import** drc**.**HtmlEntities**;**

**import** drc**.**CompilerInfo**;**

**import** drc**.**Unicode**;**

**import** drc**.**SourceText**;**

**import** drc**.**Time**;**

**import** common**;**

**import** cidrus **:** strtof**,** strtod**,** strtold**;**

**import** cidrus **:** ERANGE**;**

**import** core**.**Vararg**;**

**extern** **(**C**)** **int** getErrno**();** // for internal use

**extern** **(**C**)** **int** setErrno**(int);** // for internal use

**alias** getErrno errno**;**

**alias** setErrno errno**;**

**public** **import** drc**.**lexer**.**Funcs**;**

/// Лексер анализирует символы исходного текста и

/// производит дважды-линкованный список сем (токенов).

**class** **Лексер**

**{**

ИсходныйТекст исхТекст**;** /// Исходный текст.

сим**\*** p**;** /// Указывает на текущий символ в исходном тексте.

сим**\*** конец**;** /// Указывает на символ после конца исходного текста.

Сема**\*** глава**;** /// Глава дважды линкованного списка сем.

Сема**\*** хвост**;** /// Хвост линкованного список. Set in сканируй().

Сема**\*** сема**;** /// Указывает на текущую сему в списке сем.

// Members used for ошибка сообщения:

Диагностика диаг**;**

ОшибкаЛексера**[]** ошибки**;**

/// Всегда указывает на первый символ текущей строки.

сим**\*** началоСтроки**;**

// Сема\* нс; /// Current нс сема.

бцел номСтр **=** 1**;** /// Current, actual source текст line число.

бцел lineNum\_hline**;** /// Line число установи by #line.

бцел inTokenString**;** /// > 0 if внутри q{ }

/// Holds the original file путь and the modified one (by #line.)

ДанныеНовСтр**.**ФПути**\*** путиКФайлам**;**

/// Конструировать Лексер объект.

/// Параметры:

/// исхТекст = the UTF-8 source код.

/// диаг = used for collecting ошибка сообщения.

**this(**ИсходныйТекст исхТекст**,** Диагностика диаг **=** **null)**

**{**

**this.**исхТекст **=** исхТекст**;**

**this.**диаг **=** диаг**;**

**assert(**текст**.**length **&&** текст**[**$**-**1**]** **==** 0**,** "в исходнике отсутствует символ sentinel"**);**

**this.**p **=** текст**.**ptr**;**

**this.**конец **=** **this.**p **+** текст**.**length**;**

**this.**началоСтроки **=** **this.**p**;**

**this.**глава **=** **new** Сема**;**

**this.**глава**.**вид **=** TOK**.**ГОЛОВА**;**

**this.**глава**.**старт **=** **this.**глава**.**конец **=** **this.**p**;**

**this.**сема **=** **this.**глава**;**

// Initialize this.путиКФайлам.

новыйПутьФ**(this.**исхТекст**.**путьКФайлу**);**

// Add a нс as the first сема after the глава.

**auto** нс **=** **new** Сема**;**

нс**.**вид **=** TOK**.**Новстр**;**

нс**.**установиФлагПробельные**();**

нс**.**старт **=** нс**.**конец **=** **this.**p**;**

нс**.**нс**.**путиКФайлам **=** **this.**путиКФайлам**;**

нс**.**нс**.**oriLineNum **=** 1**;**

нс**.**нс**.**setLineNum **=** 0**;**

// Link in.

**this.**сема**.**следщ **=** нс**;**

нс**.**предш **=** **this.**сема**;**

**this.**сема **=** нс**;**

// this.нс = нс;

сканируйШебанг**();**

**}**

/// The destructor deletes the doubly-linked сема список.

**~this()**

**{**

**auto** сема **=** глава**.**следщ**;**

**while** **(**сема **!is** **null)**

**{**

**assert(**сема**.**вид **==** TOK**.**КФ **?** сема **==** хвост **&&** сема**.**следщ **is** **null** **:** 1**);**

**delete** сема**.**предш**;**

сема **=** сема**.**следщ**;**

**}**

**delete** хвост**;**

**}**

ткст текст**()**

**{**

**return** исхТекст**.**данные**;**

**}**

/// The "shebang" may optionally appear once at the beginning of a file.

/// Regexp: #![^\EndOfLine]\*

проц сканируйШебанг**()**

**{**

**if** **(\***p **==** '#' **&&** p**[**1**]** **==** '!'**)**

**{**

**auto** t **=** **new** Сема**;**

t**.**вид **=** TOK**.**Шебанг**;**

t**.**установиФлагПробельные**();**

t**.**старт **=** p**;**

**++**p**;**

**while** **(!**конецСтроки\_ли**(++**p**))**

аски\_ли**(\***p**)** **||** раскодируйЮ8**();**

t**.**конец **=** p**;**

**this.**сема**.**следщ **=** t**;**

t**.**предш **=** **this.**сема**;**

**}**

**}**

/// Sets the значение of the special сема.

проц закончиОсобуюСему**(**ref Сема t**)**

**{**

**assert(**t**.**исхТекст**[**0**..**2**]** **==** "\_\_"**);**

**switch** **(**t**.**вид**)**

**{**

**case** TOK**.**ФАЙЛ**:**

t**.**ткт **=** **this.**путиКФайлам**.**устПуть**;**

**break;**

**case** TOK**.**СТРОКА**:**

t**.**бцел\_ **=** **this.**номерСтрокиОшиб**(this.**номСтр**);**

**break;**

**case** TOK**.**ДАТА**,**

TOK**.**ВРЕМЯ**,**

TOK**.**ШТАМПВРЕМЕНИ**:**

**auto** ткт\_время **=** Время**.**вТкст**();**

**switch** **(**t**.**вид**)**

**{**

**case** TOK**.**ДАТА**:**

ткт\_время **=** Время**.**день\_месяца**(**ткт\_время**)** **~** ' ' **~** Время**.**год**(**ткт\_время**);** **break;**

**case** TOK**.**ВРЕМЯ**:**

ткт\_время **=** Время**.**время**(**ткт\_время**);** **break;**

**case** TOK**.**ШТАМПВРЕМЕНИ**:**

**break;** // ткт\_время is the timestamp.

**default:** **assert(**0**);**

**}**

ткт\_время **~=** '\0'**;** // Terminate with a zero.

t**.**ткт **=** ткт\_время**;**

**break;**

**case** TOK**.**ПОСТАВЩИК**:**

t**.**ткт **=** ПОСТАВЩИК**;**

**break;**

**case** TOK**.**ВЕРСИЯ**:**

t**.**бцел\_ **=** VERSION\_MAJOR**\***1000 **+** VERSION\_MINOR**;**

**break;**

**default:**

**assert(**0**);**

**}**

**}**

/// Sets a new file путь.

проц новыйПутьФ**(**ткст новПуть**)**

**{**

**auto** пути **=** **new** ДанныеНовСтр**.**ФПути**;**

пути**.**исхПуть **=** **this.**исхТекст**.**путьКФайлу**;**

пути**.**устПуть **=** новПуть**;**

**this.**путиКФайлам **=** пути**;**

**}**

**private** проц установиНачалоСтроки**(**сим**\*** p**)**

**{**

// Check that we can look behind one символ.

**assert((**p**-**1**)** **>=** текст**.**ptr **&&** p **<** конец**);**

// Check that предшious символ is a нс.

**assert(**конецНовСтроки\_ли**(**p **-** 1**));**

**this.**началоСтроки **=** p**;**

**}**

/// Scans the следщ сема in the source текст.

///

/// Creates a new сема if t.следщ is null and appends it в the список.

**private** проц сканируйСледщ**(**ref Сема**\*** t**)**

**{**

**assert(**t **!is** **null);**

**if** **(**t**.**следщ**)**

**{**

t **=** t**.**следщ**;**

// if (t.вид == TOK.Новстр)

// this.нс = t;

**}**

**else** **if** **(**t **!=** **this.**хвост**)**

**{**

Сема**\*** т\_нов **=** **new** Сема**;**

сканируй**(\***т\_нов**);**

т\_нов**.**предш **=** t**;**

t**.**следщ **=** т\_нов**;**

t **=** т\_нов**;**

**}**

**}**

/// Advance t one сема forward.

проц возьми**(**ref Сема**\*** t**)**

**{**

сканируйСледщ**(**t**);**

**}**

/// Advance в the следщ сема in the source текст.

TOK следщСема**()**

**{**

сканируйСледщ**(this.**сема**);**

**return** **this.**сема**.**вид**;**

**}**

/// Returns да if p points в the last символ of a Новстр.

бул конецНовСтроки\_ли**(**сим**\*** p**)**

**{**

**if** **(\***p **==** '\n' **||** **\***p **==** '\r'**)**

**return** да**;**

**if** **(\***p **==** РС**[**2**]** **||** **\***p **==** РА**[**2**])**

**if** **((**p**-**2**)** **>=** текст**.**ptr**)**

**if** **(**p**[-**1**]** **==** РС**[**1**]** **&&** p**[-**2**]** **==** РС**[**0**])**

**return** да**;**

**return** нет**;**

**}**

/// The main method which recognizes the characters that make up a сема.

///

/// Complicated семы are scanned in separate methods.

**public** проц сканируй**(**ref Сема t**)**

**in**

**{**

**assert(**текст**.**ptr **<=** p **&&** p **<** конец**);**

**}**

**out**

**{**

**assert(**текст**.**ptr **<=** t**.**старт **&&** t**.**старт **<** конец**,** Сема**.**вТкст**(**t**.**вид**));**

**assert(**текст**.**ptr **<=** t**.**конец **&&** t**.**конец **<=** конец**,** Сема**.**вТкст**(**t**.**вид**));**

**}**

**body**

**{**

// Scan whitespace.

**if** **(**пбел\_ли**(\***p**))**

**{**

t**.**пп **=** p**;**

**while** **(**пбел\_ли**(\*++**p**))**

**{}**

**}**

// Scan a сема.

бцел c **=** **\***p**;**

**{**

t**.**старт **=** p**;**

// Новстр.

**switch** **(\***p**)**

**{**

**case** '\r'**:**

**if** **(**p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**assert(**конецНовСтроки\_ли**(**p**));**

**++**p**;**

**++**номСтр**;**

установиНачалоСтроки**(**p**);**

// this.нс = &t;

t**.**вид **=** TOK**.**Новстр**;**

t**.**установиФлагПробельные**();**

t**.**нс**.**путиКФайлам **=** **this.**путиКФайлам**;**

t**.**нс**.**oriLineNum **=** номСтр**;**

t**.**нс**.**setLineNum **=** lineNum\_hline**;**

t**.**конец **=** p**;**

**return;**

**default:**

**if** **(**новСтрЮ\_ли**(**p**))**

**{**

**++**p**;** **++**p**;**

**goto** **case** '\n'**;**

**}**

**}**

// Идентификатор or ткст literal.

**if** **(**начсим\_ли**(**c**))**

**{**

**if** **(**c **==** 'r' **&&** p**[**1**]** **==** '"' **&&** **++**p**)**

**return** scanRawStringLiteral**(**t**);**

**if** **(**c **==** 'x' **&&** p**[**1**]** **==** '"'**)**

**return** scanHexStringLiteral**(**t**);**

**version(**D2**)**

**{**

**if** **(**c **==** 'q' **&&** p**[**1**]** **==** '"'**)**

**return** scanDelimitedStringLiteral**(**t**);**

**if** **(**c **==** 'q' **&&** p**[**1**]** **==** '{'**)**

**return** scanTokenStringLiteral**(**t**);**

**}**

// Scan identifier.

Lidentifier**:**

**do**

**{** c **=** **\*++**p**;** **}**

**while** **(**идент\_ли**(**c**)** **||** **!**аски\_ли**(**c**)** **&&** юАльфа\_ли**())**

t**.**конец **=** p**;**

**auto** ид **=** ТаблицаИд**.**сыщи**(**t**.**исхТекст**);**

t**.**вид **=** ид**.**вид**;**

t**.**идент **=** ид**;**

**if** **(**t**.**вид **==** TOK**.**Идентификатор **||** t**.**кслово\_ли**)**

**return;**

**else** **if** **(**t**.**спецСема\_ли**)**

закончиОсобуюСему**(**t**);**

**else** **if** **(**t**.**вид **==** TOK**.**КФ**)**

**{**

хвост **=** **&**t**;**

**assert(**t**.**исхТекст **==** "\_\_EOF\_\_"**);**

**}**

**else**

**assert(**0**,** "неожидаемый тип семы: " **~** Сема**.**вТкст**(**t**.**вид**));**

**return;**

**}**

**if** **(**цифра\_ли**(**c**))**

**return** scanNumber**(**t**);**

**if** **(**c **==** '/'**)**

**{**

c **=** **\*++**p**;**

**switch(**c**)**

**{**

**case** '='**:**

**++**p**;**

t**.**вид **=** TOK**.**ДелениеПрисвой**;**

t**.**конец **=** p**;**

**return;**

**case** '+'**:**

**return** scanNestedComment**(**t**);**

**case** '\*'**:**

**return** scanBlockComment**(**t**);**

**case** '/'**:**

**while** **(!**конецСтроки\_ли**(++**p**))**

аски\_ли**(\***p**)** **||** раскодируйЮ8**();**

t**.**вид **=** TOK**.**Комментарий**;**

t**.**установиФлагПробельные**();**

t**.**конец **=** p**;**

**return;**

**default:**

t**.**вид **=** TOK**.**Деление**;**

t**.**конец **=** p**;**

**return;**

**}**

**}**

**switch** **(**c**)**

**{**

**case** '\''**:**

**return** сканируйСимвольныйЛитерал**(**t**);**

**case** '`'**:**

**return** scanRawStringLiteral**(**t**);**

**case** '"'**:**

**return** scanNormalStringLiteral**(**t**);**

**case** '\\'**:**

ткст буфер**;**

**do**

**{**

бул isBinary**;**

c **=** scanEscapeSequence**(**isBinary**);**

**if** **(**аски\_ли**(**c**)** **||** isBinary**)**

буфер **~=** c**;**

**else**

encodeUTF8**(**буфер**,** c**);**

**}** **while** **(\***p **==** '\\'**)**

буфер **~=** 0**;**

t**.**вид **=** TOK**.**Ткст**;**

t**.**ткт **=** буфер**;**

t**.**конец **=** p**;**

**return;**

**case** '>'**:** /\* > >= >> >>= >>> >>>= \*/

c **=** **\*++**p**;**

**switch** **(**c**)**

**{**

**case** '='**:**

t**.**вид **=** TOK**.**БольшеРавно**;**

**goto** Lcommon**;**

**case** '>'**:**

**if** **(**p**[**1**]** **==** '>'**)**

**{**

**++**p**;**

**if** **(**p**[**1**]** **==** '='**)**

**{** **++**p**;**

t**.**вид **=** TOK**.**URShiftAssign**;**

**}**

**else**

t**.**вид **=** TOK**.**URShift**;**

**}**

**else** **if** **(**p**[**1**]** **==** '='**)**

**{**

**++**p**;**

t**.**вид **=** TOK**.**ПСдвигПрисвой**;**

**}**

**else**

t**.**вид **=** TOK**.**ПСдвиг**;**

**goto** Lcommon**;**

**default:**

t**.**вид **=** TOK**.**Больше**;**

**goto** Lcommon2**;**

**}**

**assert(**0**);**

**case** '<'**:** /\* < <= <> <>= << <<= \*/

c **=** **\*++**p**;**

**switch** **(**c**)**

**{**

**case** '='**:**

t**.**вид **=** TOK**.**МеньшеРавно**;**

**goto** Lcommon**;**

**case** '<'**:**

**if** **(**p**[**1**]** **==** '='**)** **{**

**++**p**;**

t**.**вид **=** TOK**.**ЛСдвигПрисвой**;**

**}**

**else**

t**.**вид **=** TOK**.**ЛСдвиг**;**

**goto** Lcommon**;**

**case** '>'**:**

**if** **(**p**[**1**]** **==** '='**)** **{**

**++**p**;**

t**.**вид **=** TOK**.**LorEorG**;**

**}**

**else**

t**.**вид **=** TOK**.**LorG**;**

**goto** Lcommon**;**

**default:**

t**.**вид **=** TOK**.**Меньше**;**

**goto** Lcommon2**;**

**}**

**assert(**0**);**

**case** '!'**:** /\* ! !< !> !<= !>= !<> !<>= \*/

c **=** **\*++**p**;**

**switch** **(**c**)**

**{**

**case** '<'**:**

c **=** **\*++**p**;**

**if** **(**c **==** '>'**)**

**{**

**if** **(**p**[**1**]** **==** '='**)** **{**

**++**p**;**

t**.**вид **=** TOK**.**Unordered**;**

**}**

**else**

t**.**вид **=** TOK**.**UorE**;**

**}**

**else** **if** **(**c **==** '='**)**

**{**

t**.**вид **=** TOK**.**UorG**;**

**}**

**else** **{**

t**.**вид **=** TOK**.**UorGorE**;**

**goto** Lcommon2**;**

**}**

**goto** Lcommon**;**

**case** '>'**:**

**if** **(**p**[**1**]** **==** '='**)**

**{**

**++**p**;**

t**.**вид **=** TOK**.**UorL**;**

**}**

**else**

t**.**вид **=** TOK**.**UorLorE**;**

**goto** Lcommon**;**

**case** '='**:**

t**.**вид **=** TOK**.**НеРавно**;**

**goto** Lcommon**;**

**default:**

t**.**вид **=** TOK**.**Не**;**

**goto** Lcommon2**;**

**}**

**assert(**0**);**

**case** '.'**:** /\* . .[0-9] .. ... \*/

**if** **(**p**[**1**]** **==** '.'**)**

**{**

**++**p**;**

**if** **(**p**[**1**]** **==** '.'**)** **{**

**++**p**;**

t**.**вид **=** TOK**.**Эллипсис**;**

**}**

**else**

t**.**вид **=** TOK**.**Срез**;**

**}**

**else** **if** **(**цифра\_ли**(**p**[**1**]))**

**{**

**return** scanReal**(**t**);**

**}**

**else**

t**.**вид **=** TOK**.**Точка**;**

**goto** Lcommon**;**

**case** '|'**:** /\* | || |= \*/

c **=** **\*++**p**;**

**if** **(**c **==** '='**)**

t**.**вид **=** TOK**.**ИлиПрисвой**;**

**else** **if** **(**c **==** '|'**)**

t**.**вид **=** TOK**.**ИлиЛогическое**;**

**else** **{**

t**.**вид **=** TOK**.**ИлиБинарное**;**

**goto** Lcommon2**;**

**}**

**goto** Lcommon**;**

**case** '&'**:** /\* & && &= \*/

c **=** **\*++**p**;**

**if** **(**c **==** '='**)**

t**.**вид **=** TOK**.**ИПрисвой**;**

**else** **if** **(**c **==** '&'**)**

t**.**вид **=** TOK**.**ИЛогическое**;**

**else** **{**

t**.**вид **=** TOK**.**ИБинарное**;**

**goto** Lcommon2**;**

**}**

**goto** Lcommon**;**

**case** '+'**:** /\* + ++ += \*/

c **=** **\*++**p**;**

**if** **(**c **==** '='**)**

t**.**вид **=** TOK**.**ПлюсПрисвой**;**

**else** **if** **(**c **==** '+'**)**

t**.**вид **=** TOK**.**ПлюсПлюс**;**

**else** **{**

t**.**вид **=** TOK**.**Плюс**;**

**goto** Lcommon2**;**

**}**

**goto** Lcommon**;**

**case** '-'**:** /\* - -- -= \*/

c **=** **\*++**p**;**

**if** **(**c **==** '='**)**

t**.**вид **=** TOK**.**МинусПрисвой**;**

**else** **if** **(**c **==** '-'**)**

t**.**вид **=** TOK**.**МинусМинус**;**

**else** **{**

t**.**вид **=** TOK**.**Минус**;**

**goto** Lcommon2**;**

**}**

**goto** Lcommon**;**

**case** '='**:** /\* = == \*/

**if** **(**p**[**1**]** **==** '='**)** **{**

**++**p**;**

t**.**вид **=** TOK**.**Равно**;**

**}**

**else**

t**.**вид **=** TOK**.**Присвоить**;**

**goto** Lcommon**;**

**case** '~'**:** /\* ~ ~= \*/

**if** **(**p**[**1**]** **==** '='**)** **{**

**++**p**;**

t**.**вид **=** TOK**.**CatAssign**;**

**}**

**else**

t**.**вид **=** TOK**.**Тильда**;**

**goto** Lcommon**;**

**case** '\*'**:** /\* \* \*= \*/

**if** **(**p**[**1**]** **==** '='**)** **{**

**++**p**;**

t**.**вид **=** TOK**.**УмножьПрисвой**;**

**}**

**else**

t**.**вид **=** TOK**.**Умножь**;**

**goto** Lcommon**;**

**case** '^'**:** /\* ^ ^= \*/

**if** **(**p**[**1**]** **==** '='**)** **{**

**++**p**;**

t**.**вид **=** TOK**.**ИИлиПрисвой**;**

**}**

**else**

t**.**вид **=** TOK**.**ИИли**;**

**goto** Lcommon**;**

**case** '%'**:** /\* % %= \*/

**if** **(**p**[**1**]** **==** '='**)** **{**

**++**p**;**

t**.**вид **=** TOK**.**МодульПрисвой**;**

**}**

**else**

t**.**вид **=** TOK**.**Модуль**;**

**goto** Lcommon**;**

// Single символ семы:

**case** '('**:**

t**.**вид **=** TOK**.**ЛСкобка**;**

**goto** Lcommon**;**

**case** ')'**:**

t**.**вид **=** TOK**.**ПСкобка**;**

**goto** Lcommon**;**

**case** '['**:**

t**.**вид **=** TOK**.**ЛКвСкобка**;**

**goto** Lcommon**;**

**case** ']'**:**

t**.**вид **=** TOK**.**ПКвСкобка**;**

**goto** Lcommon**;**

**case** '{'**:**

t**.**вид **=** TOK**.**ЛФСкобка**;**

**goto** Lcommon**;**

**case** '}'**:**

t**.**вид **=** TOK**.**ПФСкобка**;**

**goto** Lcommon**;**

**case** ':'**:**

t**.**вид **=** TOK**.**Двоеточие**;**

**goto** Lcommon**;**

**case** ';'**:**

t**.**вид **=** TOK**.**ТочкаЗапятая**;**

**goto** Lcommon**;**

**case** '?'**:**

t**.**вид **=** TOK**.**Вопрос**;**

**goto** Lcommon**;**

**case** ','**:**

t**.**вид **=** TOK**.**Запятая**;**

**goto** Lcommon**;**

**case** '$'**:**

t**.**вид **=** TOK**.**Доллар**;**

Lcommon**:**

**++**p**;**

Lcommon2**:**

t**.**конец **=** p**;**

**return;**

**case** '#'**:**

**return** scanSpecialTokenSequence**(**t**);**

**default:**

**}**

// Check for КФ

**if** **(**кф\_ли**(**c**))**

**{**

**assert(**кф\_ли**(\***p**),** ""**~\***p**);**

t**.**вид **=** TOK**.**КФ**;**

t**.**конец **=** p**;**

хвост **=** **&**t**;**

**assert(**t**.**старт **==** t**.**конец**);**

**return;**

**}**

**if** **(!**аски\_ли**(**c**))**

**{**

c **=** раскодируйЮ8**();**

**if** **(**униАльфа\_ли**(**c**))**

**goto** Lidentifier**;**

**}**

ошибка**(**t**.**старт**,** ИДС**.**НедопустимыйСимвол**,** **cast(**дим**)**c**);**

**++**p**;**

t**.**вид **=** TOK**.**Нелегал**;**

t**.**установиФлагПробельные**();**

t**.**дим\_ **=** c**;**

t**.**конец **=** p**;**

**return;**

**}**

**}**

/// Converts a ткст literal в an integer.

**template** toБцел**(**ткст T**)**

**{**

**static** **assert(**0 **<** T**.**length **&&** T**.**length **<=** 4**);**

**static** **if** **(**T**.**length **==** 1**)**

**const** бцел toБцел **=** T**[**0**];**

**else**

**const** бцел toБцел **=** **(**T**[**0**]** **<<** **((**T**.**length**-**1**)\***8**))** **|** toБцел**!(**T**[**1**..**$**]);**

**}**

**static** **assert(**toБцел**!(**"\xAA\xBB\xCC\xDD"**)** **==** 0xAABBCCDD**);**

/// Constructs case инструкции. E.g.:

/// ---

//// // case\_!("<", "Меньше", "Lcommon") ->

/// case 60u:

/// t.вид = TOK.Меньше;

/// goto Lcommon;

/// ---

/// FIXME: Can't use this yet due в a $(DMDBUG 1534, bug) in DMD.

**template** case\_**(**ткст ткт**,** ткст вид**,** ткст лейбл**)**

**{**

**const** ткст case\_ **=**

`case `**~**toБцел**!(**ткт**).**stringof**~**`:`

`t.вид = TOK.`**~**вид**~**`;`

`goto `**~**лейбл**~**`;`**;**

**}**

//pragma(сооб, case\_!("<", "Меньше", "Lcommon"));

**template** case\_L4**(**ткст ткт**,** TOK вид**)**

**{**

**const** ткст case\_L4 **=** case\_**!(**ткт**,** вид**,** "Lcommon\_4"**);**

**}**

**template** case\_L3**(**ткст ткт**,** TOK вид**)**

**{**

**const** ткст case\_L3 **=** case\_**!(**ткт**,** вид**,** "Lcommon\_3"**);**

**}**

**template** case\_L2**(**ткст ткт**,** TOK вид**)**

**{**

**const** ткст case\_L2 **=** case\_**!(**ткт**,** вид**,** "Lcommon\_2"**);**

**}**

**template** case\_L1**(**ткст ткт**,** TOK вид**)**

**{**

**const** ткст case\_L3 **=** case\_**!(**ткт**,** вид**,** "Lcommon"**);**

**}**

/// An alternative сканируй method.

/// Profiling shows it's a bit slower.

**public** проц scan\_**(**ref Сема t**)**

**in**

**{**

**assert(**текст**.**ptr **<=** p **&&** p **<** конец**);**

**}**

**out**

**{**

**assert(**текст**.**ptr **<=** t**.**старт **&&** t**.**старт **<** конец**,** Сема**.**вТкст**(**t**.**вид**));**

**assert(**текст**.**ptr **<=** t**.**конец **&&** t**.**конец **<=** конец**,** Сема**.**вТкст**(**t**.**вид**));**

**}**

**body**

**{**

// Scan whitespace.

**if** **(**пбел\_ли**(\***p**))**

**{**

t**.**пп **=** p**;**

**while** **(**пбел\_ли**(\*++**p**))**

**{}**

**}**

// Scan a сема.

t**.**старт **=** p**;**

// Новстр.

**switch** **(\***p**)**

**{**

**case** '\r'**:**

**if** **(**p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**assert(**конецНовСтроки\_ли**(**p**));**

**++**p**;**

**++**номСтр**;**

установиНачалоСтроки**(**p**);**

// this.нс = &t;

t**.**вид **=** TOK**.**Новстр**;**

t**.**установиФлагПробельные**();**

t**.**нс**.**путиКФайлам **=** **this.**путиКФайлам**;**

t**.**нс**.**oriLineNum **=** номСтр**;**

t**.**нс**.**setLineNum **=** lineNum\_hline**;**

t**.**конец **=** p**;**

**return;**

**default:**

**if** **(**новСтрЮ\_ли**(**p**))**

**{**

**++**p**;** **++**p**;**

**goto** **case** '\n'**;**

**}**

**}**

бцел c **=** **\***p**;**

**assert(**конец **-** p **!=** 0**);**

**switch** **(**конец **-** p**)**

**{**

**case** 1**:**

**goto** L1character**;**

**case** 2**:**

c **<<=** 8**;** c **|=** p**[**1**];**

**goto** L2characters**;**

**case** 3**:**

c **<<=** 8**;** c **|=** p**[**1**];** c **<<=** 8**;** c **|=** p**[**2**];**

**goto** L3characters**;**

**default:**

**version(**BigEndian**)**

c **=** **\*cast(**бцел**\*)**p**;**

**else**

**{**

c **<<=** 8**;** c **|=** p**[**1**];** c **<<=** 8**;** c **|=** p**[**2**];** c **<<=** 8**;** c **|=** p**[**3**];**

/+

c = \*cast(бцел\*)p;

asm

{

mov EDX, c;

bswap EDX;

mov c, EDX;

}

+/

**}**

**}**

// 4 символ семы.

**switch** **(**c**)**

**{**

**case** toБцел**!(**">>>="**):**

t**.**вид **=** TOK**.**ПСдвигПрисвой**;**

**goto** Lcommon\_4**;**

**case** toБцел**!(**"!<>="**):**

t**.**вид **=** TOK**.**Unordered**;**

Lcommon\_4**:**

p **+=** 4**;**

t**.**конец **=** p**;**

**return;**

**default:**

**}**

c **>>>=** 8**;**

L3characters**:**

**assert(**p **==** t**.**старт**);**

// 3 символ семы.

**switch** **(**c**)**

**{**

**case** toБцел**!(**">>="**):**

t**.**вид **=** TOK**.**ПСдвигПрисвой**;**

**goto** Lcommon\_3**;**

**case** toБцел**!(**">>>"**):**

t**.**вид **=** TOK**.**URShift**;**

**goto** Lcommon\_3**;**

**case** toБцел**!(**"<>="**):**

t**.**вид **=** TOK**.**LorEorG**;**

**goto** Lcommon\_3**;**

**case** toБцел**!(**"<<="**):**

t**.**вид **=** TOK**.**ЛСдвигПрисвой**;**

**goto** Lcommon\_3**;**

**case** toБцел**!(**"!<="**):**

t**.**вид **=** TOK**.**UorG**;**

**goto** Lcommon\_3**;**

**case** toБцел**!(**"!>="**):**

t**.**вид **=** TOK**.**UorL**;**

**goto** Lcommon\_3**;**

**case** toБцел**!(**"!<>"**):**

t**.**вид **=** TOK**.**UorE**;**

**goto** Lcommon\_3**;**

**case** toБцел**!(**"..."**):**

t**.**вид **=** TOK**.**Эллипсис**;**

Lcommon\_3**:**

p **+=** 3**;**

t**.**конец **=** p**;**

**return;**

**default:**

**}**

c **>>>=** 8**;**

L2characters**:**

**assert(**p **==** t**.**старт**);**

// 2 символ семы.

**switch** **(**c**)**

**{**

**case** toБцел**!(**"/+"**):**

**++**p**;** // Skip /

**return** scanNestedComment**(**t**);**

**case** toБцел**!(**"/\*"**):**

**++**p**;** // Skip /

**return** scanBlockComment**(**t**);**

**case** toБцел**!(**"//"**):**

**++**p**;** // Skip /

**assert(\***p **==** '/'**);**

**while** **(!**конецСтроки\_ли**(++**p**))**

аски\_ли**(\***p**)** **||** раскодируйЮ8**();**

t**.**вид **=** TOK**.**Комментарий**;**

t**.**установиФлагПробельные**();**

t**.**конец **=** p**;**

**return;**

**case** toБцел**!(**">="**):**

t**.**вид **=** TOK**.**БольшеРавно**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**">>"**):**

t**.**вид **=** TOK**.**ПСдвиг**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"<<"**):**

t**.**вид **=** TOK**.**ЛСдвиг**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"<="**):**

t**.**вид **=** TOK**.**МеньшеРавно**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"<>"**):**

t**.**вид **=** TOK**.**LorG**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"!<"**):**

t**.**вид **=** TOK**.**UorGorE**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"!>"**):**

t**.**вид **=** TOK**.**UorLorE**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"!="**):**

t**.**вид **=** TOK**.**НеРавно**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**".."**):**

t**.**вид **=** TOK**.**Срез**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"&&"**):**

t**.**вид **=** TOK**.**ИЛогическое**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"&="**):**

t**.**вид **=** TOK**.**ИПрисвой**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"||"**):**

t**.**вид **=** TOK**.**ИлиЛогическое**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"|="**):**

t**.**вид **=** TOK**.**ИлиПрисвой**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"++"**):**

t**.**вид **=** TOK**.**ПлюсПлюс**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"+="**):**

t**.**вид **=** TOK**.**ПлюсПрисвой**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"--"**):**

t**.**вид **=** TOK**.**МинусМинус**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"-="**):**

t**.**вид **=** TOK**.**МинусПрисвой**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"=="**):**

t**.**вид **=** TOK**.**Равно**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"~="**):**

t**.**вид **=** TOK**.**CatAssign**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"\*="**):**

t**.**вид **=** TOK**.**УмножьПрисвой**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"/="**):**

t**.**вид **=** TOK**.**ДелениеПрисвой**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"^="**):**

t**.**вид **=** TOK**.**ИИлиПрисвой**;**

**goto** Lcommon\_2**;**

**case** toБцел**!(**"%="**):**

t**.**вид **=** TOK**.**МодульПрисвой**;**

Lcommon\_2**:**

p **+=** 2**;**

t**.**конец **=** p**;**

**return;**

**default:**

**}**

c **>>>=** 8**;**

L1character**:**

**assert(**p **==** t**.**старт**);**

**assert(\***p **==** c**,** Формат**(**"p={0},c={1}"**,** **\***p**,** **cast(**дим**)**c**));**

// 1 символ семы.

// TODO: conсторонаr storing the сема тип in ptable.

**switch** **(**c**)**

**{**

**case** '\''**:**

**return** сканируйСимвольныйЛитерал**(**t**);**

**case** '`'**:**

**return** scanRawStringLiteral**(**t**);**

**case** '"'**:**

**return** scanNormalStringLiteral**(**t**);**

**case** '\\'**:**

ткст буфер**;**

**do**

**{**

бул isBinary**;**

c **=** scanEscapeSequence**(**isBinary**);**

**if** **(**аски\_ли**(**c**)** **||** isBinary**)**

буфер **~=** c**;**

**else**

encodeUTF8**(**буфер**,** c**);**

**}** **while** **(\***p **==** '\\'**)**

буфер **~=** 0**;**

t**.**вид **=** TOK**.**Ткст**;**

t**.**ткт **=** буфер**;**

t**.**конец **=** p**;**

**return;**

**case** '<'**:**

t**.**вид **=** TOK**.**Больше**;**

**goto** Lcommon**;**

**case** '>'**:**

t**.**вид **=** TOK**.**Меньше**;**

**goto** Lcommon**;**

**case** '^'**:**

t**.**вид **=** TOK**.**ИИли**;**

**goto** Lcommon**;**

**case** '!'**:**

t**.**вид **=** TOK**.**Не**;**

**goto** Lcommon**;**

**case** '.'**:**

**if** **(**цифра\_ли**(**p**[**1**]))**

**return** scanReal**(**t**);**

t**.**вид **=** TOK**.**Точка**;**

**goto** Lcommon**;**

**case** '&'**:**

t**.**вид **=** TOK**.**ИБинарное**;**

**goto** Lcommon**;**

**case** '|'**:**

t**.**вид **=** TOK**.**ИлиБинарное**;**

**goto** Lcommon**;**

**case** '+'**:**

t**.**вид **=** TOK**.**Плюс**;**

**goto** Lcommon**;**

**case** '-'**:**

t**.**вид **=** TOK**.**Минус**;**

**goto** Lcommon**;**

**case** '='**:**

t**.**вид **=** TOK**.**Присвоить**;**

**goto** Lcommon**;**

**case** '~'**:**

t**.**вид **=** TOK**.**Тильда**;**

**goto** Lcommon**;**

**case** '\*'**:**

t**.**вид **=** TOK**.**Умножь**;**

**goto** Lcommon**;**

**case** '/'**:**

t**.**вид **=** TOK**.**Деление**;**

**goto** Lcommon**;**

**case** '%'**:**

t**.**вид **=** TOK**.**Модуль**;**

**goto** Lcommon**;**

**case** '('**:**

t**.**вид **=** TOK**.**ЛСкобка**;**

**goto** Lcommon**;**

**case** ')'**:**

t**.**вид **=** TOK**.**ПСкобка**;**

**goto** Lcommon**;**

**case** '['**:**

t**.**вид **=** TOK**.**ЛКвСкобка**;**

**goto** Lcommon**;**

**case** ']'**:**

t**.**вид **=** TOK**.**ПКвСкобка**;**

**goto** Lcommon**;**

**case** '{'**:**

t**.**вид **=** TOK**.**ЛФСкобка**;**

**goto** Lcommon**;**

**case** '}'**:**

t**.**вид **=** TOK**.**ПФСкобка**;**

**goto** Lcommon**;**

**case** ':'**:**

t**.**вид **=** TOK**.**Двоеточие**;**

**goto** Lcommon**;**

**case** ';'**:**

t**.**вид **=** TOK**.**ТочкаЗапятая**;**

**goto** Lcommon**;**

**case** '?'**:**

t**.**вид **=** TOK**.**Вопрос**;**

**goto** Lcommon**;**

**case** ','**:**

t**.**вид **=** TOK**.**Запятая**;**

**goto** Lcommon**;**

**case** '$'**:**

t**.**вид **=** TOK**.**Доллар**;**

Lcommon**:**

**++**p**;**

t**.**конец **=** p**;**

**return;**

**case** '#'**:**

**return** scanSpecialTokenSequence**(**t**);**

**default:**

**}**

**assert(**p **==** t**.**старт**);**

**assert(\***p **==** c**);**

// TODO: conсторонаr moving начсим\_ли() and цифра\_ли() up.

**if** **(**начсим\_ли**(**c**))**

**{**

**if** **(**c **==** 'r' **&&** p**[**1**]** **==** '"' **&&** **++**p**)**

**return** scanRawStringLiteral**(**t**);**

**if** **(**c **==** 'x' **&&** p**[**1**]** **==** '"'**)**

**return** scanHexStringLiteral**(**t**);**

**version(**D2**)**

**{**

**if** **(**c **==** 'q' **&&** p**[**1**]** **==** '"'**)**

**return** scanDelimitedStringLiteral**(**t**);**

**if** **(**c **==** 'q' **&&** p**[**1**]** **==** '{'**)**

**return** scanTokenStringLiteral**(**t**);**

**}**

// Scan identifier.

Lidentifier**:**

**do**

**{** c **=** **\*++**p**;** **}**

**while** **(**идент\_ли**(**c**)** **||** **!**аски\_ли**(**c**)** **&&** юАльфа\_ли**())**

t**.**конец **=** p**;**

**auto** ид **=** ТаблицаИд**.**сыщи**(**t**.**исхТекст**);**

t**.**вид **=** ид**.**вид**;**

t**.**идент **=** ид**;**

**if** **(**t**.**вид **==** TOK**.**Идентификатор **||** t**.**кслово\_ли**)**

**return;**

**else** **if** **(**t**.**спецСема\_ли**)**

закончиОсобуюСему**(**t**);**

**else** **if** **(**t**.**вид **==** TOK**.**КФ**)**

**{**

хвост **=** **&**t**;**

**assert(**t**.**исхТекст **==** "\_\_EOF\_\_"**);**

**}**

**else**

**assert(**0**,** "unexpected сема тип: " **~** Сема**.**вТкст**(**t**.**вид**));**

**return;**

**}**

**if** **(**цифра\_ли**(**c**))**

**return** scanNumber**(**t**);**

// Check for КФ

**if** **(**кф\_ли**(**c**))**

**{**

**assert(**кф\_ли**(\***p**),** **\***p**~**""**);**

t**.**вид **=** TOK**.**КФ**;**

t**.**конец **=** p**;**

хвост **=** **&**t**;**

**assert(**t**.**старт **==** t**.**конец**);**

**return;**

**}**

**if** **(!**аски\_ли**(**c**))**

**{**

c **=** раскодируйЮ8**();**

**if** **(**униАльфа\_ли**(**c**))**

**goto** Lidentifier**;**

**}**

ошибка**(**t**.**старт**,** ИДС**.**НедопустимыйСимвол**,** **cast(**дим**)**c**);**

**++**p**;**

t**.**вид **=** TOK**.**Нелегал**;**

t**.**установиФлагПробельные**();**

t**.**дим\_ **=** c**;**

t**.**конец **=** p**;**

**return;**

**}**

/// Scans a block comment.

///

/// BlockComment := "/\*" AnyChar\* "\*/"

проц scanBlockComment**(**ref Сема t**)**

**{**

**assert(**p**[-**1**]** **==** '/' **&&** **\***p **==** '\*'**);**

**auto** tokenLineNum **=** номСтр**;**

**auto** tokenLineBegin **=** началоСтроки**;**

Loop**:**

**while** **(**1**)**

**{**

**switch** **(\*++**p**)**

**{**

**case** '\*'**:**

**if** **(**p**[**1**]** **!=** '/'**)**

**continue;**

p **+=** 2**;**

**break** Loop**;**

**case** '\r'**:**

**if** **(**p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**assert(**конецНовСтроки\_ли**(**p**));**

**++**номСтр**;**

установиНачалоСтроки**(**p**+**1**);**

**break;**

**default:**

**if** **(!**аски\_ли**(\***p**))**

**{**

**if** **(**симНовСтрЮ\_ли**(**раскодируйЮ8**()))**

**goto** **case** '\n'**;**

**}**

**else** **if** **(**кф\_ли**(\***p**))**

**{**

ошибка**(**tokenLineNum**,** tokenLineBegin**,** t**.**старт**,** ИДС**.**UnterminatedBlockComment**);**

**break** Loop**;**

**}**

**}**

**}**

t**.**вид **=** TOK**.**Комментарий**;**

t**.**установиФлагПробельные**();**

t**.**конец **=** p**;**

**return;**

**}**

/// Scans a nested comment.

///

/// NestedComment := "/+" (AnyChar\* | NestedComment) "+/"

проц scanNestedComment**(**ref Сема t**)**

**{**

**assert(**p**[-**1**]** **==** '/' **&&** **\***p **==** '+'**);**

**auto** tokenLineNum **=** номСтр**;**

**auto** tokenLineBegin **=** началоСтроки**;**

бцел уровень **=** 1**;**

Loop**:**

**while** **(**1**)**

**{**

**switch** **(\*++**p**)**

**{**

**case** '/'**:**

**if** **(**p**[**1**]** **==** '+'**)**

**++**p**,** **++**уровень**;**

**continue;**

**case** '+'**:**

**if** **(**p**[**1**]** **!=** '/'**)**

**continue;**

**++**p**;**

**if** **(--**уровень **!=** 0**)**

**continue;**

**++**p**;**

**break** Loop**;**

**case** '\r'**:**

**if** **(**p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**assert(**конецНовСтроки\_ли**(**p**));**

**++**номСтр**;**

установиНачалоСтроки**(**p**+**1**);**

**continue;**

**default:**

**if** **(!**аски\_ли**(\***p**))**

**{**

**if** **(**симНовСтрЮ\_ли**(**раскодируйЮ8**()))**

**goto** **case** '\n'**;**

**}**

**else** **if** **(**кф\_ли**(\***p**))**

**{**

ошибка**(**tokenLineNum**,** tokenLineBegin**,** t**.**старт**,** ИДС**.**UnterminatedNestedComment**);**

**break** Loop**;**

**}**

**}**

**}**

t**.**вид **=** TOK**.**Комментарий**;**

t**.**установиФлагПробельные**();**

t**.**конец **=** p**;**

**return;**

**}**

/// Scans the postfix символ of a ткст literal.

///

/// PostfixChar := "c" | "w" | "d"

сим scanPostfix**()**

**{**

**assert(**p**[-**1**]** **==** '"' **||** p**[-**1**]** **==** '`' **||**

**{** **version(**D2**)** **return** p**[-**1**]** **==** '}'**;**

**else** **return** 0**;** **}()**

**);**

**switch** **(\***p**)**

**{**

**case** 'c'**:**

**case** 'w'**:**

**case** 'd'**:**

**return** **\***p**++;**

**default:**

**return** 0**;**

**}**

**assert(**0**);**

**}**

/// Scans a normal ткст literal.

///

/// NormalStringLiteral := "\"" Сим\* "\""

проц scanNormalStringLiteral**(**ref Сема t**)**

**{**

**assert(\***p **==** '"'**);**

**auto** tokenLineNum **=** номСтр**;**

**auto** tokenLineBegin **=** началоСтроки**;**

t**.**вид **=** TOK**.**Ткст**;**

ткст буфер**;**

бцел c**;**

**while** **(**1**)**

**{**

c **=** **\*++**p**;**

**switch** **(**c**)**

**{**

**case** '"'**:**

**++**p**;**

t**.**pf **=** scanPostfix**();**

Lreturn**:**

t**.**ткт **=** буфер **~** '\0'**;**

t**.**конец **=** p**;**

**return;**

**case** '\\'**:**

бул isBinary**;**

c **=** scanEscapeSequence**(**isBinary**);**

**--**p**;**

**if** **(**аски\_ли**(**c**)** **||** isBinary**)**

буфер **~=** c**;**

**else**

encodeUTF8**(**буфер**,** c**);**

**continue;**

**case** '\r'**:**

**if** **(**p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**assert(**конецНовСтроки\_ли**(**p**));**

c **=** '\n'**;** // Convert Новстр в \n.

**++**номСтр**;**

установиНачалоСтроки**(**p**+**1**);**

**break;**

**case** 0**,** \_Z\_**:**

ошибка**(**tokenLineNum**,** tokenLineBegin**,** t**.**старт**,** ИДС**.**НеоконченыйТкст**);**

**goto** Lreturn**;**

**default:**

**if** **(!**аски\_ли**(**c**))**

**{**

c **=** раскодируйЮ8**();**

**if** **(**симНовСтрЮ\_ли**(**c**))**

**goto** **case** '\n'**;**

encodeUTF8**(**буфер**,** c**);**

**continue;**

**}**

**}**

**assert(**аски\_ли**(**c**));**

буфер **~=** c**;**

**}**

**assert(**0**);**

**}**

/// Scans a символ literal.

///

/// СимЛитерал := "'" Сим "'"

проц сканируйСимвольныйЛитерал**(**ref Сема t**)**

**{**

**assert(\***p **==** '\''**);**

**++**p**;**

t**.**вид **=** TOK**.**СимЛитерал**;**

**switch** **(\***p**)**

**{**

**case** '\\'**:**

бул notused**;**

t**.**дим\_ **=** scanEscapeSequence**(**notused**);**

**break;**

**case** '\''**:**

ошибка**(**t**.**старт**,** ИДС**.**ПустойСимвольныйЛитерал**);**

**break;**

**default:**

**if** **(**конецСтроки\_ли**(**p**))**

**break;**

бцел c **=** **\***p**;**

**if** **(!**аски\_ли**(**c**))**

c **=** раскодируйЮ8**();**

t**.**дим\_ **=** c**;**

**++**p**;**

**}**

**if** **(\***p **==** '\''**)**

**++**p**;**

**else**

ошибка**(**t**.**старт**,** ИДС**.**НеоконченныйСимвольныйЛитерал**);**

t**.**конец **=** p**;**

**}**

/// Scans a raw ткст literal.

///

/// RawStringLiteral := "r\"" AnyChar\* "\"" | "`" AnyChar\* "`"

проц scanRawStringLiteral**(**ref Сема t**)**

**{**

**assert(\***p **==** '`' **||** **\***p **==** '"' **&&** p**[-**1**]** **==** 'r'**);**

**auto** tokenLineNum **=** номСтр**;**

**auto** tokenLineBegin **=** началоСтроки**;**

t**.**вид **=** TOK**.**Ткст**;**

бцел delim **=** **\***p**;**

ткст буфер**;**

бцел c**;**

**while** **(**1**)**

**{**

c **=** **\*++**p**;**

**switch** **(**c**)**

**{**

**case** '\r'**:**

**if** **(**p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**assert(**конецНовСтроки\_ли**(**p**));**

c **=** '\n'**;** // Convert Новстр в '\n'.

**++**номСтр**;**

установиНачалоСтроки**(**p**+**1**);**

**break;**

**case** '`'**:**

**case** '"'**:**

**if** **(**c **==** delim**)**

**{**

**++**p**;**

t**.**pf **=** scanPostfix**();**

Lreturn**:**

t**.**ткт **=** буфер **~** '\0'**;**

t**.**конец **=** p**;**

**return;**

**}**

**break;**

**case** 0**,** \_Z\_**:**

ошибка**(**tokenLineNum**,** tokenLineBegin**,** t**.**старт**,**

delim **==** 'r' **?** ИДС**.**UnterminatedRawString **:** ИДС**.**UnterminatedBackQuoteString**);**

**goto** Lreturn**;**

**default:**

**if** **(!**аски\_ли**(**c**))**

**{**

c **=** раскодируйЮ8**();**

**if** **(**симНовСтрЮ\_ли**(**c**))**

**goto** **case** '\n'**;**

encodeUTF8**(**буфер**,** c**);**

**continue;**

**}**

**}**

**assert(**аски\_ли**(**c**));**

буфер **~=** c**;**

**}**

**assert(**0**);**

**}**

/// Scans a hexadecimal ткст literal.

///

/// HexStringLiteral := "x\"" (HexChar HexChar)\* "\""

проц scanHexStringLiteral**(**ref Сема t**)**

**{**

**assert(**p**[**0**]** **==** 'x' **&&** p**[**1**]** **==** '"'**);**

t**.**вид **=** TOK**.**Ткст**;**

**auto** tokenLineNum **=** номСтр**;**

**auto** tokenLineBegin **=** началоСтроки**;**

бцел c**;**

ббайт**[]** буфер**;**

ббайт h**;** // hex число

бцел n**;** // число of hex digits

**++**p**;**

**assert(\***p **==** '"'**);**

**while** **(**1**)**

**{**

c **=** **\*++**p**;**

**switch** **(**c**)**

**{**

**case** '"'**:**

**if** **(**n **&** 1**)**

ошибка**(**tokenLineNum**,** tokenLineBegin**,** t**.**старт**,** ИДС**.**OddNumberOfDigitsInHexString**);**

**++**p**;**

t**.**pf **=** scanPostfix**();**

Lreturn**:**

t**.**ткт **=** **cast(**ткст**)** **(**буфер **~=** 0**);**

t**.**конец **=** p**;**

**return;**

**case** '\r'**:**

**if** **(**p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**assert(**конецНовСтроки\_ли**(**p**));**

**++**номСтр**;**

установиНачалоСтроки**(**p**+**1**);**

**continue;**

**default:**

**if** **(**гекс\_ли**(**c**))**

**{**

**if** **(**c **<=** '9'**)**

c **-=** '0'**;**

**else** **if** **(**c **<=** 'F'**)**

c **-=** 'A' **-** 10**;**

**else**

c **-=** 'a' **-** 10**;**

**if** **(**n **&** 1**)**

**{**

h **<<=** 4**;**

h **|=** c**;**

буфер **~=** h**;**

**}**

**else**

h **=** **cast(**ббайт**)**c**;**

**++**n**;**

**continue;**

**}**

**else** **if** **(**пбел\_ли**(**c**))**

**continue;** // Skip spaces.

**else** **if** **(**кф\_ли**(**c**))**

**{**

ошибка**(**tokenLineNum**,** tokenLineBegin**,** t**.**старт**,** ИДС**.**UnterminatedHexString**);**

t**.**pf **=** 0**;**

**goto** Lreturn**;**

**}**

**else**

**{**

**auto** errorAt **=** p**;**

**if** **(!**аски\_ли**(**c**))**

**{**

c **=** раскодируйЮ8**();**

**if** **(**симНовСтрЮ\_ли**(**c**))**

**goto** **case** '\n'**;**

**}**

ошибка**(**errorAt**,** ИДС**.**NonHexCharInHexString**,** **cast(**дим**)**c**);**

**}**

**}**

**}**

**assert(**0**);**

**}**

**version(**DDoc**)**

**{**

/// Scans a delimited ткст literal.

проц scanDelimitedStringLiteral**(**ref Сема t**);**

/// Scans a сема ткст literal.

///

/// TokenStringLiteral := "q{" Сема\* "}"

проц scanTokenStringLiteral**(**ref Сема t**);**

**}**

**else**

**version(**D2**)**

**{**

проц scanDelimitedStringLiteral**(**ref Сема t**)**

**{**

**assert(**p**[**0**]** **==** 'q' **&&** p**[**1**]** **==** '"'**);**

t**.**вид **=** TOK**.**Ткст**;**

**auto** tokenLineNum **=** номСтр**;**

**auto** tokenLineBegin **=** началоСтроки**;**

ткст буфер**;**

дим открывающий\_delim **=** 0**,** // 0 if no nested delimiter or '[', '(', '<', '{'

закрывающий\_delim**;** // Will be ']', ')', '>', '},

// the first символ of an identifier or

// any другой Unicode/ASCII символ.

ткст ткт\_delim**;** // Идентификатор delimiter.

бцел уровень **=** 1**;** // Counter for nestable delimiters.

**++**p**;** **++**p**;** // Skip q"

бцел c **=** **\***p**;**

**switch** **(**c**)**

**{**

**case** '('**:**

открывающий\_delim **=** c**;**

закрывающий\_delim **=** ')'**;** // c + 1

**break;**

**case** '['**,** '<'**,** '{'**:**

открывающий\_delim **=** c**;**

закрывающий\_delim **=** c **+** 2**;** // Get в закрывающий counterpart. Feature of ASCII таблица.

**break;**

**default:**

дим сканируйНовСтр**()**

**{**

**switch** **(\***p**)**

**{**

**case** '\r'**:**

**if** **(**p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**assert(**конецНовСтроки\_ли**(**p**));**

**++**p**;**

**++**номСтр**;**

установиНачалоСтроки**(**p**);**

**break;**

**default:**

**if** **(**новСтрЮ\_ли**(**p**))** **{**

p **+=** 2**;**

**goto** **case** '\n'**;**

**}**

**return** нет**;**

**}**

**return** да**;**

**}**

// Skip leading newlines:

**while** **(**сканируйНовСтр**())**

**{}**

**assert(!**новСтр\_ли**(**p**));**

сим**\*** начало **=** p**;**

c **=** **\***p**;**

закрывающий\_delim **=** c**;**

// TODO: Check for non-printable characters?

**if** **(!**аски\_ли**(**c**))**

**{**

закрывающий\_delim **=** раскодируйЮ8**();**

**if** **(!**униАльфа\_ли**(**закрывающий\_delim**))**

**break;** // Не an identifier.

**}**

**else** **if** **(!**начсим\_ли**(**c**))**

**break;** // Не an identifier.

// Parse Идентификатор + EndOfLine

**do**

**{** c **=** **\*++**p**;** **}**

**while** **(**идент\_ли**(**c**)** **||** **!**аски\_ли**(**c**)** **&&** юАльфа\_ли**())**

// Store identifier

ткт\_delim **=** начало**[**0**..**p**-**начало**];**

// Scan нс

**if** **(**сканируйНовСтр**())**

**--**p**;** // Go back one because of "c = \*++p;" in main loop.

**else**

**{**

// TODO: ошибка(p, ИДС.ExpectedNewlineAfterIdentDelim);

**}**

**}**

бул checkStringDelim**(**сим**\*** p**)**

**{**

**assert(**ткт\_delim**.**length **!=** 0**);**

**if** **(**буфер**[**$**-**1**]** **==** '\n' **&&** // Last символ copied в буфер must be '\n'.

конец**-**p **>=** ткт\_delim**.**length **&&** // Check remaining length.

p**[**0**..**ткт\_delim**.**length**]** **==** ткт\_delim**)** // Compare.

**return** да**;**

**return** нет**;**

**}**

**while** **(**1**)**

**{**

c **=** **\*++**p**;**

**switch** **(**c**)**

**{**

**case** '\r'**:**

**if** **(**p**[**1**]** **==** '\n'**)**

**++**p**;**

**case** '\n'**:**

**assert(**конецНовСтроки\_ли**(**p**));**

c **=** '\n'**;** // Convert Новстр в '\n'.

**++**номСтр**;**

установиНачалоСтроки**(**p**+**1**);**

**break;**

**case** 0**,** \_Z\_**:**

// TODO: ошибка(tokenLineNum, tokenLineBegin, t.старт, ИДС.UnterminatedDelimitedString);

**goto** Lreturn3**;**

**default:**

**if** **(!**аски\_ли**(**c**))**

**{**

**auto** начало **=** p**;**

c **=** раскодируйЮ8**();**

**if** **(**симНовСтрЮ\_ли**(**c**))**

**goto** **case** '\n'**;**

**if** **(**c **==** закрывающий\_delim**)**

**{**

**if** **(**ткт\_delim**.**length**)**

**{**

**if** **(**checkStringDelim**(**начало**))**

**{**

p **=** начало **+** ткт\_delim**.**length**;**

**goto** Lreturn2**;**

**}**

**}**

**else**

**{**

**assert(**уровень **==** 1**);**

**--**уровень**;**

**goto** Lreturn**;**

**}**

**}**

encodeUTF8**(**буфер**,** c**);**

**continue;**

**}**

**else**

**{**

**if** **(**c **==** открывающий\_delim**)**

**++**уровень**;**

**else** **if** **(**c **==** закрывающий\_delim**)**

**{**

**if** **(**ткт\_delim**.**length**)**

**{**

**if** **(**checkStringDelim**(**p**))**

**{**

p **+=** ткт\_delim**.**length**;**

**goto** Lreturn2**;**

**}**

**}**

**else** **if** **(--**уровень **==** 0**)**

**goto** Lreturn**;**

**}**

**}**

**}**

**assert(**аски\_ли**(**c**));**

буфер **~=** c**;**

**}**

Lreturn**:** // Character delimiter.

**assert(**c **==** закрывающий\_delim**);**

**assert(**уровень **==** 0**);**

**++**p**;** // Skip закрывающий delimiter.

Lreturn2**:** // Ткст delimiter.

**if** **(\***p **==** '"'**)**

**++**p**;**

**else**

**{**

// TODO: ошибка(p, ИДС.ExpectedDblQuoteAfterDelim, ткт\_delim.length ? ткт\_delim : закрывающий\_delim~"");

**}**

t**.**pf **=** scanPostfix**();**

Lreturn3**:** // Ошибка.

t**.**ткт **=** буфер **~** '\0'**;**

t**.**конец **=** p**;**

**}**

проц scanTokenStringLiteral**(**ref Сема t**)**

**{**

**assert(**p**[**0**]** **==** 'q' **&&** p**[**1**]** **==** '{'**);**

t**.**вид **=** TOK**.**Ткст**;**

**auto** tokenLineNum **=** номСтр**;**

**auto** tokenLineBegin **=** началоСтроки**;**

// A guard against changes в particular члены:

// this.lineNum\_hline and this.errorPath

**++**inTokenString**;**

бцел номСтр **=** **this.**номСтр**;**

бцел уровень **=** 1**;**

**++**p**;** **++**p**;** // Skip q{

**auto** предш\_t **=** **&**t**;**

Сема**\*** сема**;**

**while** **(**1**)**

**{**

сема **=** **new** Сема**;**

сканируй**(\***сема**);**

// Save the семы in a doubly linked список.

// Could be useful for various tools.

сема**.**предш **=** предш\_t**;**

предш\_t**.**следщ **=** сема**;**

предш\_t **=** сема**;**

**switch** **(**сема**.**вид**)**

**{**

**case** TOK**.**ЛФСкобка**:**

**++**уровень**;**

**continue;**

**case** TOK**.**ПФСкобка**:**

**if** **(--**уровень **==** 0**)**

**{**

t**.**tok\_ткт **=** t**.**следщ**;**

t**.**следщ **=** **null;**

**break;**

**}**

**continue;**

**case** TOK**.**КФ**:**

// TODO: ошибка(tokenLineNum, tokenLineBegin, t.старт, ИДС.UnterminatedTokenString);

t**.**tok\_ткт **=** t**.**следщ**;**

t**.**следщ **=** сема**;**

**break;**

**default:**

**continue;**

**}**

**break;** // Exit loop.

**}**

**assert(**сема**.**вид **==** TOK**.**ПФСкобка **||** сема**.**вид **==** TOK**.**КФ**);**

**assert(**сема**.**вид **==** TOK**.**ПФСкобка **&&** t**.**следщ **is** **null** **||**

сема**.**вид **==** TOK**.**КФ **&&** t**.**следщ **!is** **null);**

ткст буфер**;**

// сема points в } or КФ

**if** **(**сема**.**вид **==** TOK**.**КФ**)**

**{**

t**.**конец **=** сема**.**старт**;**

буфер **=** t**.**исхТекст**[**2**..**$**].**dup **~** '\0'**;**

**}**

**else**

**{**

// Присвоить в буфер before scanPostfix().

t**.**конец **=** p**;**

буфер **=** t**.**исхТекст**[**2**..**$**-**1**].**dup **~** '\0'**;**

t**.**pf **=** scanPostfix**();**

t**.**конец **=** p**;** // Присвоить again because of postfix.

**}**

// Convert newlines в '\n'.

**if** **(**номСтр **!=** **this.**номСтр**)**

**{**

**assert(**буфер**[**$**-**1**]** **==** '\0'**);**

бцел i**,** j**;**

**for** **(;** i **<** буфер**.**length**;** **++**i**)**

**switch** **(**буфер**[**i**])**

**{**

**case** '\r'**:**

**if** **(**буфер**[**i**+**1**]** **==** '\n'**)**

**++**i**;**

**case** '\n'**:**

**assert(**конецНовСтроки\_ли**(**буфер**.**ptr **+** i**));**

буфер**[**j**++]** **=** '\n'**;** // Convert Новстр в '\n'.

**break;**

**default:**

**if** **(**новСтрЮ\_ли**(**буфер**.**ptr **+** i**))**

**{**

**++**i**;** **++**i**;**

**goto** **case** '\n'**;**

**}**

буфер**[**j**++]** **=** буфер**[**i**];** // Copy.

**}**

буфер**.**length **=** j**;** // Adjust length.

**}**

**assert(**буфер**[**$**-**1**]** **==** '\0'**);**

t**.**ткт **=** буфер**;**

**--**inTokenString**;**

**}**

**}** // version(D2)

/// Scans an escape sequence.

///

/// EscapeSequence := "\" (Восмиричный{1,3} | ("x" Гекс{2}) |

/// ("u" Гекс{4}) | ("U" Гекс{8}) |

/// "'" | "\"" | "\\" | "?" | "a" |

/// "b" | "f" | "n" | "r" | "t" | "v")

/// Параметры:

/// isBinary = установи в да for octal and hexadecimal escapes.

/// Возвращает: the escape значение.

дим scanEscapeSequence**(**ref бул isBinary**)**

**out(**результат**)**

**{** **assert(**верноСимвол\_ли**(**результат**));** **}**

**body**

**{**

**assert(\***p **==** '\\'**);**

**auto** sequenceStart **=** p**;** // Used for ошибка reporting.

**++**p**;**

бцел c **=** сим8еск**(\***p**);**

**if** **(**c**)**

**{**

**++**p**;**

**return** c**;**

**}**

бцел digits **=** 2**;**

**switch** **(\***p**)**

**{**

**case** 'x'**:**

isBinary **=** да**;**

case\_Unicode**:**

**assert(**c **==** 0**);**

**assert(**digits **==** 2 **||** digits **==** 4 **||** digits **==** 8**);**

**while** **(**1**)**

**{**

**++**p**;**

**if** **(**гекс\_ли**(\***p**))**

**{**

c **\*=** 16**;**

**if** **(\***p **<=** '9'**)**

c **+=** **\***p **-** '0'**;**

**else** **if** **(\***p **<=** 'F'**)**

c **+=** **\***p **-** 'A' **+** 10**;**

**else**

c **+=** **\***p **-** 'a' **+** 10**;**

**if** **(--**digits **==** 0**)**

**{**

**++**p**;**

**if** **(**верноСимвол\_ли**(**c**))**

**return** c**;** // Итог valid escape значение.

ошибка**(**sequenceStart**,** ИДС**.**InvalidUnicodeEscapeSequence**,**

sequenceStart**[**0**..**p**-**sequenceStart**]);**

**break;**

**}**

**continue;**

**}**

ошибка**(**sequenceStart**,** ИДС**.**InsufficientHexDigits**,**

sequenceStart**[**0**..**p**-**sequenceStart**]);**

**break;**

**}**

**break;**

**case** 'u'**:**

digits **=** 4**;**

**goto** case\_Unicode**;**

**case** 'U'**:**

digits **=** 8**;**

**goto** case\_Unicode**;**

**default:**

**if** **(**восмир\_ли**(\***p**))**

**{**

isBinary **=** да**;**

**assert(**c **==** 0**);**

c **+=** **\***p **-** '0'**;**

**++**p**;**

**if** **(!**восмир\_ли**(\***p**))**

**return** c**;**

c **\*=** 8**;**

c **+=** **\***p **-** '0'**;**

**++**p**;**

**if** **(!**восмир\_ли**(\***p**))**

**return** c**;**

c **\*=** 8**;**

c **+=** **\***p **-** '0'**;**

**++**p**;**

**if** **(**c **>** 0xFF**)**

ошибка**(**sequenceStart**,** сооб**.**НевернаяВосмеричнаяПоследовательностьУклонения**,**

sequenceStart**[**0**..**p**-**sequenceStart**]);**

**return** c**;** // Итог valid escape значение.

**}**

**else** **if(\***p **==** '&'**)**

**{**

**if** **(**буква\_ли**(\*++**p**))**

**{**

**auto** начало **=** p**;**

**while** **(**цифробукв\_ли**(\*++**p**))**

**{}**

**if** **(\***p **==** ';'**)**

**{**

// Pass сущность excluding '&' and ';'.

c **=** сущностьВЮникод**(**начало**[**0**..**p **-** начало**]);**

**++**p**;** // Skip ;

**if** **(**c **!=** 0xFFFF**)**

**return** c**;** // Итог valid escape значение.

**else**

ошибка**(**sequenceStart**,** ИДС**.**UndefinedHTMLEntity**,** sequenceStart**[**0 **..** p **-** sequenceStart**]);**

**}**

**else**

ошибка**(**sequenceStart**,** ИДС**.**UnterminatedHTMLEntity**,** sequenceStart**[**0 **..** p **-** sequenceStart**]);**

**}**

**else**

ошибка**(**sequenceStart**,** ИДС**.**InvalidBeginHTMLEntity**);**

**}**

**else** **if** **(**конецСтроки\_ли**(**p**))**

ошибка**(**sequenceStart**,** ИДС**.**UndefinedEscapeSequence**,**

кф\_ли**(\***p**)** **?** `\КФ` **:** `\NewLine`**);**

**else**

**{**

ткст ткт **=** `\`**;**

**if** **(**аски\_ли**(**c**))**

ткт **~=** **\***p**;**

**else**

encodeUTF8**(**ткт**,** раскодируйЮ8**());**

**++**p**;**

// TODO: check for unprintable символ?

ошибка**(**sequenceStart**,** ИДС**.**UndefinedEscapeSequence**,** ткт**);**

**}**

**}**

**return** СИМ\_ЗАМЕНЫ**;** // Ошибка: return replacement символ.

**}**

/// Scans a число literal.

///

/// $(PRE

/// IntegerLiteral := (Dec|Гекс|Bin|Oct)Suffix?

/// Dec := (0|[1-9][0-9\_]\*)

/// Гекс := 0[xX][\_]\*[0-9a-zA-Z][0-9a-zA-Z\_]\*

/// Bin := 0[bB][\_]\*[01][01\_]\*

/// Oct := 0[0-7\_]\*

/// Suffix := (L[uU]?|[uU]L?)

/// )

/// Неверно: "0b\_", "0x\_", ".\_" etc.

проц scanNumber**(**ref Сема t**)**

**{**

бдол бдол\_**;**

бул overflow**;**

бул isDecimal**;**

т\_мера digits**;**

**if** **(\***p **!=** '0'**)**

**goto** LscanInteger**;**

**++**p**;** // пропусти zero

// check for xX bB ...

**switch** **(\***p**)**

**{**

**case** 'x'**,**'X'**:**

**goto** LscanHex**;**

**case** 'b'**,**'B'**:**

**goto** LscanBinary**;**

**case** 'L'**:**

**if** **(**p**[**1**]** **==** 'i'**)**

**goto** LscanReal**;** // 0Li

**break;** // 0L

**case** '.'**:**

**if** **(**p**[**1**]** **==** '.'**)**

**break;** // 0..

// 0.

**case** 'i'**,**'f'**,**'F'**,** // Мнимое and плав literal suffixes.

'e'**,** 'E'**:** // Плав exponent.

**goto** LscanReal**;**

**default:**

**if** **(\***p **==** '\_'**)**

**goto** LscanOctal**;** // 0\_

**else** **if** **(**цифра\_ли**(\***p**))**

**{**

**if** **(\***p **==** '8' **||** **\***p **==** '9'**)**

**goto** Loctal\_hasDecimalDigits**;** // 08 or 09

**else**

**goto** Loctal\_enter\_loop**;** // 0[0-7]

**}**

**}**

// Число 0

**assert(**p**[-**1**]** **==** '0'**);**

**assert(\***p **!=** '\_' **&&** **!**цифра\_ли**(\***p**));**

**assert(**бдол\_ **==** 0**);**

isDecimal **=** да**;**

**goto** Lfinalize**;**

LscanInteger**:**

**assert(\***p **!=** 0 **&&** цифра\_ли**(\***p**));**

isDecimal **=** да**;**

**goto** Lenter\_loop\_int**;**

**while** **(**1**)**

**{**

**if** **(\*++**p **==** '\_'**)**

**continue;**

**if** **(!**цифра\_ли**(\***p**))**

**break;**

Lenter\_loop\_int**:**

**if** **(**бдол\_ **<** бдол**.**max**/**10 **||** **(**бдол\_ **==** бдол**.**max**/**10 **&&** **\***p **<=** '5'**))**

**{**

бдол\_ **\*=** 10**;**

бдол\_ **+=** **\***p **-** '0'**;**

**continue;**

**}**

// Overflow: пропусти following digits.

overflow **=** да**;**

**while** **(**цифра\_ли**(\*++**p**))** **{}**

**break;**

**}**

// The число could be a плав, so check overflow below.

**switch** **(\***p**)**

**{**

**case** '.'**:**

**if** **(**p**[**1**]** **!=** '.'**)**

**goto** LscanReal**;**

**break;**

**case** 'L'**:**

**if** **(**p**[**1**]** **!=** 'i'**)**

**break;**

**case** 'i'**,** 'f'**,** 'F'**,** 'e'**,** 'E'**:**

**goto** LscanReal**;**

**default:**

**}**

**if** **(**overflow**)**

ошибка**(**t**.**старт**,** ИДС**.**OverflowDecimalNumber**);**

**assert((**цифра\_ли**(**p**[-**1**])** **||** p**[-**1**]** **==** '\_'**)** **&&** **!**цифра\_ли**(\***p**)** **&&** **\***p **!=** '\_'**);**

**goto** Lfinalize**;**

LscanHex**:**

**assert(**digits **==** 0**);**

**assert(\***p **==** 'x' **||** **\***p **==** 'X'**);**

**while** **(**1**)**

**{**

**if** **(\*++**p **==** '\_'**)**

**continue;**

**if** **(!**гекс\_ли**(\***p**))**

**break;**

**++**digits**;**

бдол\_ **\*=** 16**;**

**if** **(\***p **<=** '9'**)**

бдол\_ **+=** **\***p **-** '0'**;**

**else** **if** **(\***p **<=** 'F'**)**

бдол\_ **+=** **\***p **-** 'A' **+** 10**;**

**else**

бдол\_ **+=** **\***p **-** 'a' **+** 10**;**

**}**

**assert(**гекс\_ли**(**p**[-**1**])** **||** p**[-**1**]** **==** '\_' **||** p**[-**1**]** **==** 'x' **||** p**[-**1**]** **==** 'X'**);**

**assert(!**гекс\_ли**(\***p**)** **&&** **\***p **!=** '\_'**);**

**switch** **(\***p**)**

**{**

**case** '.'**:**

**if** **(**p**[**1**]** **==** '.'**)**

**break;**

**case** 'p'**,** 'P'**:**

**return** scanHexReal**(**t**);**

**default:**

**}**

**if** **(**digits **==** 0 **||** digits **>** 16**)**

ошибка**(**t**.**старт**,** digits **==** 0 **?** ИДС**.**NoDigitsInHexNumber **:** ИДС**.**OverflowHexNumber**);**

**goto** Lfinalize**;**

LscanBinary**:**

**assert(**digits **==** 0**);**

**assert(\***p **==** 'b' **||** **\***p **==** 'B'**);**

**while** **(**1**)**

**{**

**if** **(\*++**p **==** '0'**)**

**{**

**++**digits**;**

бдол\_ **\*=** 2**;**

**}**

**else** **if** **(\***p **==** '1'**)**

**{**

**++**digits**;**

бдол\_ **\*=** 2**;**

бдол\_ **+=** **\***p **-** '0'**;**

**}**

**else** **if** **(\***p **==** '\_'**)**

**continue;**

**else**

**break;**

**}**

**if** **(**digits **==** 0 **||** digits **>** 64**)**

ошибка**(**t**.**старт**,** digits **==** 0 **?** ИДС**.**NoDigitsInBinNumber **:** ИДС**.**OverflowBinaryNumber**);**

**assert(**p**[-**1**]** **==** '0' **||** p**[-**1**]** **==** '1' **||** p**[-**1**]** **==** '\_' **||** p**[-**1**]** **==** 'b' **||** p**[-**1**]** **==** 'B'**,** p**[-**1**]** **~** ""**);**

**assert(** **!(\***p **==** '0' **||** **\***p **==** '1' **||** **\***p **==** '\_'**)** **);**

**goto** Lfinalize**;**

LscanOctal**:**

**assert(\***p **==** '\_'**);**

**while** **(**1**)**

**{**

**if** **(\*++**p **==** '\_'**)**

**continue;**

**if** **(!**восмир\_ли**(\***p**))**

**break;**

Loctal\_enter\_loop**:**

**if** **(**бдол\_ **<** бдол**.**max**/**2 **||** **(**бдол\_ **==** бдол**.**max**/**2 **&&** **\***p **<=** '1'**))**

**{**

бдол\_ **\*=** 8**;**

бдол\_ **+=** **\***p **-** '0'**;**

**continue;**

**}**

// Overflow: пропусти following digits.

overflow **=** да**;**

**while** **(**восмир\_ли**(\*++**p**))** **{}**

**break;**

**}**

бул hasDecimalDigits**;**

**if** **(**цифра\_ли**(\***p**))**

**{**

Loctal\_hasDecimalDigits**:**

hasDecimalDigits **=** да**;**

**while** **(**цифра\_ли**(\*++**p**))** **{}**

**}**

// The число could be a плав, so check ошибки below.

**switch** **(\***p**)**

**{**

**case** '.'**:**

**if** **(**p**[**1**]** **!=** '.'**)**

**goto** LscanReal**;**

**break;**

**case** 'L'**:**

**if** **(**p**[**1**]** **!=** 'i'**)**

**break;**

**case** 'i'**,** 'f'**,** 'F'**,** 'e'**,** 'E'**:**

**goto** LscanReal**;**

**default:**

**}**

**if** **(**hasDecimalDigits**)**

ошибка**(**t**.**старт**,** ИДС**.**OctalNumberHasDecimals**);**

**if** **(**overflow**)**

ошибка**(**t**.**старт**,** ИДС**.**OverflowOctalNumber**);**

// goto Lfinalize;

Lfinalize**:**

**enum** Suffix

**{**

Нет **=** 0**,**

Unsigned **=** 1**,**

Дол **=** 2

**}**

// Scan optional суффикс: L, Lu, LU, u, uL, U or UL.

Suffix суффикс**;**

**while** **(**1**)**

**{**

**switch** **(\***p**)**

**{**

**case** 'L'**:**

**if** **(**суффикс **&** Suffix**.**Дол**)**

**break;**

суффикс **|=** Suffix**.**Дол**;**

**++**p**;**

**continue;**

**case** 'u'**,** 'U'**:**

**if** **(**суффикс **&** Suffix**.**Unsigned**)**

**break;**

суффикс **|=** Suffix**.**Unsigned**;**

**++**p**;**

**continue;**

**default:**

**break;**

**}**

**break;**

**}**

// Determine тип of Integer.

**switch** **(**суффикс**)**

**{**

**case** Suffix**.**Нет**:**

**if** **(**бдол\_ **&** 0x8000\_0000\_0000\_0000**)**

**{**

**if** **(**isDecimal**)**

ошибка**(**t**.**старт**,** ИДС**.**OverflowDecimalSign**);**

t**.**вид **=** TOK**.**Бцел64**;**

**}**

**else** **if** **(**бдол\_ **&** 0xFFFF\_FFFF\_0000\_0000**)**

t**.**вид **=** TOK**.**Цел64**;**

**else** **if** **(**бдол\_ **&** 0x8000\_0000**)**

t**.**вид **=** isDecimal **?** TOK**.**Цел64 **:** TOK**.**Бцел32**;**

**else**

t**.**вид **=** TOK**.**Цел32**;**

**break;**

**case** Suffix**.**Unsigned**:**

**if** **(**бдол\_ **&** 0xFFFF\_FFFF\_0000\_0000**)**

t**.**вид **=** TOK**.**Бцел64**;**

**else**

t**.**вид **=** TOK**.**Бцел32**;**

**break;**

**case** Suffix**.**Дол**:**

**if** **(**бдол\_ **&** 0x8000\_0000\_0000\_0000**)**

**{**

**if** **(**isDecimal**)**

ошибка**(**t**.**старт**,** ИДС**.**OverflowDecimalSign**);**

t**.**вид **=** TOK**.**Бцел64**;**

**}**

**else**

t**.**вид **=** TOK**.**Цел64**;**

**break;**

**case** Suffix**.**Unsigned **|** Suffix**.**Дол**:**

t**.**вид **=** TOK**.**Бцел64**;**

**break;**

**default:**

**assert(**0**);**

**}**

t**.**бдол\_ **=** бдол\_**;**

t**.**конец **=** p**;**

**return;**

LscanReal**:**

scanReal**(**t**);**

**return;**

**}**

/// Scans a floating point число literal.

///

/// $(PRE

/// ПлавLiteral := Плав[fFL]?i?

/// Плав := DecПлав | HexПлав

/// DecПлав := ([0-9][0-9\_]\*[.][0-9\_]\*DecExponent?) |

/// [.][0-9][0-9\_]\*DecExponent? | [0-9][0-9\_]\*DecExponent

/// DecExponent := [eE][+-]?[0-9][0-9\_]\*

/// HexПлав := 0[xX](HexDigits[.]HexDigits |

/// [.][0-9a-zA-Z]HexDigits? |

/// HexDigits)HexExponent

/// HexExponent := [pP][+-]?[0-9][0-9\_]\*

/// )

проц scanReal**(**ref Сема t**)**

**{**

**if** **(\***p **==** '.'**)**

**{**

**assert(**p**[**1**]** **!=** '.'**);**

// Этот function was called by сканируй() or scanNumber().

**while** **(**цифра\_ли**(\*++**p**)** **||** **\***p **==** '\_'**)** **{}**

**}**

**else**

// Этот function was called by scanNumber().

**assert(delegate** **()**

**{**

**switch** **(\***p**)**

**{**

**case** 'L'**:**

**if** **(**p**[**1**]** **!=** 'i'**)**

**return** нет**;**

**case** 'i'**,** 'f'**,** 'F'**,** 'e'**,** 'E'**:**

**return** да**;**

**default:**

**}**

**return** нет**;**

**}()**

**);**

// Scan exponent.

**if** **(\***p **==** 'e' **||** **\***p **==** 'E'**)**

**{**

**++**p**;**

**if** **(\***p **==** '-' **||** **\***p **==** '+'**)**

**++**p**;**

**if** **(**цифра\_ли**(\***p**))**

**while** **(**цифра\_ли**(\*++**p**)** **||** **\***p **==** '\_'**)** **{}**

**else**

ошибка**(**t**.**старт**,** ИДС**.**ПлавExpMustStartWithDigit**);**

**}**

// Copy whole число and remove underscores из буфер.

ткст буфер **=** t**.**старт**[**0**..**p**-**t**.**старт**].**dup**;**

бцел j**;**

**foreach** **(**c**;** буфер**)**

**if** **(**c **!=** '\_'**)**

буфер**[**j**++]** **=** c**;**

буфер**.**length **=** j**;** // Adjust length.

буфер **~=** 0**;** // Terminate for C functions.

finalizeПлав**(**t**,** буфер**);**

**}**

/// Scans a hexadecimal floating point число literal.

проц scanHexReal**(**ref Сема t**)**

**{**

**assert(\***p **==** '.' **||** **\***p **==** 'p' **||** **\***p **==** 'P'**);**

ИДС идс**;**

**if** **(\***p **==** '.'**)**

**while** **(**гекс\_ли**(\*++**p**)** **||** **\***p **==** '\_'**)**

**{}**

// Decimal exponent is required.

**if** **(\***p **!=** 'p' **&&** **\***p **!=** 'P'**)**

**{**

идс **=** ИДС**.**HexПлавExponentRequired**;**

**goto** Lerr**;**

**}**

// Scan exponent

**assert(\***p **==** 'p' **||** **\***p **==** 'P'**);**

**++**p**;**

**if** **(\***p **==** '+' **||** **\***p **==** '-'**)**

**++**p**;**

**if** **(!**цифра\_ли**(\***p**))**

**{**

идс **=** ИДС**.**HexПлавExpMustStartWithDigit**;**

**goto** Lerr**;**

**}**

**while** **(**цифра\_ли**(\*++**p**)** **||** **\***p **==** '\_'**)**

**{}**

// Copy whole число and remove underscores из буфер.

ткст буфер **=** t**.**старт**[**0**..**p**-**t**.**старт**].**dup**;**

бцел j**;**

**foreach** **(**c**;** буфер**)**

**if** **(**c **!=** '\_'**)**

буфер**[**j**++]** **=** c**;**

буфер**.**length **=** j**;** // Adjust length.

буфер **~=** 0**;** // Terminate for C functions.

finalizeПлав**(**t**,** буфер**);**

**return;**

Lerr**:**

t**.**вид **=** TOK**.**Плав32**;**

t**.**конец **=** p**;**

ошибка**(**t**.**старт**,** идс**);**

**}**

/// Sets the значение of the сема.

/// Параметры:

/// t = receives the значение.

/// буфер = the well-formed плав число.

проц finalizeПлав**(**ref Сема t**,** ткст буфер**)**

**{**

**assert(**буфер**[**$**-**1**]** **==** 0**);**

// Плав число is well-formed. Check suffixes and do conversion.

**switch** **(\***p**)**

**{**

**case** 'f'**,** 'F'**:**

t**.**вид **=** TOK**.**Плав32**;**

t**.**плав\_ **=** strtof**(**буфер**.**ptr**,** **null);**

**++**p**;**

**break;**

**case** 'L'**:**

t**.**вид **=** TOK**.**Плав80**;**

t**.**реал\_ **=** strtold**(**буфер**.**ptr**,** **null);**

**++**p**;**

**break;**

**default:**

t**.**вид **=** TOK**.**Плав64**;**

t**.**дво\_ **=** strtod**(**буфер**.**ptr**,** **null);**

**}**

**if** **(\***p **==** 'i'**)**

**{**

**++**p**;**

t**.**вид **+=** 3**;** // Щит в imaginary counterpart.

**assert(**t**.**вид **==** TOK**.**Мнимое32 **||**

t**.**вид **==** TOK**.**Мнимое64 **||**

t**.**вид **==** TOK**.**Мнимое80**);**

**}**

**if** **(**errno**()** **==** ERANGE**)**

ошибка**(**t**.**старт**,** ИДС**.**OverflowПлавNumber**);**

t**.**конец **=** p**;**

**}**

/// Scans a special сема sequence.

///

/// SpecialTokenSequence := "#line" Integer Filespec? EndOfLine

проц scanSpecialTokenSequence**(**ref Сема t**)**

**{**

**assert(\***p **==** '#'**);**

t**.**вид **=** TOK**.**HashLine**;**

t**.**установиФлагПробельные**();**

ИДС идс**;**

сим**\*** errorAtColumn **=** p**;**

сим**\*** tokenEnd **=** **++**p**;**

**if** **(!(**p**[**0**]** **==** 'l' **&&** p**[**1**]** **==** 'i' **&&** p**[**2**]** **==** 'n' **&&** p**[**3**]** **==** 'e'**))**

**{**

идс **=** ИДС**.**ExpectedIdentifierSTLine**;**

**goto** Lerr**;**

**}**

p **+=** 3**;**

tokenEnd **=** p **+** 1**;**

// TODO: #line58"путь/file" is legal. Require spaces?

// State.Space could be used for that purpose.

**enum** State

**{** /+Space,+/ Integer**,** Filespec**,** End **}**

State state **=** State**.**Integer**;**

**while** **(!**конецСтроки\_ли**(++**p**))**

**{**

**if** **(**пбел\_ли**(\***p**))**

**continue;**

**if** **(**state **==** State**.**Integer**)**

**{**

**if** **(!**цифра\_ли**(\***p**))**

**{**

errorAtColumn **=** p**;**

идс **=** ИДС**.**ExpectedIntegerAfterSTLine**;**

**goto** Lerr**;**

**}**

t**.**tokLineNum **=** **new** Сема**;**

сканируй**(\***t**.**tokLineNum**);**

tokenEnd **=** p**;**

**if** **(**t**.**tokLineNum**.**вид **!=** TOK**.**Цел32 **&&** t**.**tokLineNum**.**вид **!=** TOK**.**Бцел32**)**

**{**

errorAtColumn **=** t**.**tokLineNum**.**старт**;**

идс **=** ИДС**.**ExpectedIntegerAfterSTLine**;**

**goto** Lerr**;**

**}**

**--**p**;** // Go one back because сканируй() advanced p past the integer.

state **=** State**.**Filespec**;**

**}**

**else** **if** **(**state **==** State**.**Filespec **&&** **\***p **==** '"'**)**

**{** // ИДС.ExpectedFilespec is deprecated.

// if (\*p != '"')

// {

// errorAtColumn = p;

// идс = ИДС.ExpectedFilespec;

// goto Lerr;

// }

t**.**tokLineFilespec **=** **new** Сема**;**

t**.**tokLineFilespec**.**старт **=** p**;**

t**.**tokLineFilespec**.**вид **=** TOK**.**Filespec**;**

t**.**tokLineFilespec**.**установиФлагПробельные**();**

**while** **(\*++**p **!=** '"'**)**

**{**

**if** **(**конецСтроки\_ли**(**p**))**

**{**

errorAtColumn **=** t**.**tokLineFilespec**.**старт**;**

идс **=** ИДС**.**НеоконченоеУказаниеФайла**;**

t**.**tokLineFilespec**.**конец **=** p**;**

tokenEnd **=** p**;**

**goto** Lerr**;**

**}**

аски\_ли**(\***p**)** **||** раскодируйЮ8**();**

**}**

**auto** старт **=** t**.**tokLineFilespec**.**старт **+**1**;** // +1 пропустиs '"'

t**.**tokLineFilespec**.**ткт **=** старт**[**0 **..** p **-** старт**];**

t**.**tokLineFilespec**.**конец **=** p **+** 1**;**

tokenEnd **=** p **+** 1**;**

state **=** State**.**End**;**

**}**

**else**/+ if (state == State.End)+/

**{**

идс **=** ИДС**.**НеоконченыйОсобыйТокен**;**

**goto** Lerr**;**

**}**

**}**

**assert(**конецСтроки\_ли**(**p**));**

**if** **(**state **==** State**.**Integer**)**

**{**

errorAtColumn **=** p**;**

идс **=** ИДС**.**ExpectedIntegerAfterSTLine**;**

**goto** Lerr**;**

**}**

// Evaluate #line only when not in сема ткст.

**if** **(!**inTokenString **&&** t**.**tokLineNum**)**

**{**

**this.**lineNum\_hline **=** **this.**номСтр **-** t**.**tokLineNum**.**бцел\_ **+** 1**;**

**if** **(**t**.**tokLineFilespec**)**

новыйПутьФ**(**t**.**tokLineFilespec**.**ткт**);**

**}**

p **=** tokenEnd**;**

t**.**конец **=** tokenEnd**;**

**return;**

Lerr**:**

p **=** tokenEnd**;**

t**.**конец **=** tokenEnd**;**

ошибка**(**errorAtColumn**,** идс**);**

**}**

/// Inserts an empty dummy сема (TOK.Пусто) before t.

///

/// Useful in the parsing phase for representing a узел in the AST

/// that doesn't consume an actual сема из the source текст.

Сема**\*** вставьПустуюСемуПеред**(**Сема**\*** t**)**

**{**

**assert(**t **!is** **null** **&&** t**.**предш **!is** **null);**

**assert(**текст**.**ptr **<=** t**.**старт **&&** t**.**старт **<** конец**,** Сема**.**вТкст**(**t**.**вид**));**

**assert(**текст**.**ptr **<=** t**.**конец **&&** t**.**конец **<=** конец**,** Сема**.**вТкст**(**t**.**вид**));**

**auto** предш\_t **=** t**.**предш**;**

**auto** т\_нов **=** **new** Сема**;**

т\_нов**.**вид **=** TOK**.**Пусто**;**

т\_нов**.**старт **=** т\_нов**.**конец **=** предш\_t**.**конец**;**

// Link in new сема.

предш\_t**.**следщ **=** т\_нов**;**

т\_нов**.**предш **=** предш\_t**;**

т\_нов**.**следщ **=** t**;**

t**.**предш **=** т\_нов**;**

**return** т\_нов**;**

**}**

/// Возвращает ошибка line число.

бцел номерСтрокиОшиб**(**бцел номСтр**)**

**{**

**return** номСтр **-** **this.**lineNum\_hline**;**

**}**

/// Forwards ошибка параметры.

проц ошибка**(**сим**\*** columnPos**,** ткст сооб**,** **...)**

**{**

error\_**(this.**номСтр**,** **this.**началоСтроки**,** columnPos**,** сооб**,** \_arguments**,** \_argptr**);**

**}**

/// определено

проц ошибка**(**сим**\*** columnPos**,** ИДС идс**,** **...)**

**{**

error\_**(this.**номСтр**,** **this.**началоСтроки**,** columnPos**,** ДайСооб**(**идс**),** \_arguments**,** \_argptr**);**

**}**

/// определено

проц ошибка**(**бцел номСтр**,** сим**\*** началоСтроки**,** сим**\*** columnPos**,** ИДС идс**,** **...)**

**{**

error\_**(**номСтр**,** началоСтроки**,** columnPos**,** ДайСооб**(**идс**),** \_arguments**,** \_argptr**);**

**}**

/// Creates an ошибка report and appends it в a список.

/// Параметры:

/// номСтр = the line число.

/// началоСтроки = points в the first символ of the current line.

/// columnPos = points в the символ where the ошибка is located.

/// сооб = the сообщение.

проц error\_**(**бцел номСтр**,** сим**\*** началоСтроки**,** сим**\*** columnPos**,** ткст сооб**,**

TypeInfo**[]** \_arguments**,** base**.**спис\_ва \_argptr**)**

**{**

номСтр **=** **this.**номерСтрокиОшиб**(**номСтр**);**

**auto** errorPath **=** **this.**путиКФайлам**.**устПуть**;**

**auto** положение **=** **new** Положение**(**errorPath**,** номСтр**,** началоСтроки**,** columnPos**);**

сооб **=** Формат**(**\_arguments**,** \_argptr**,** сооб**);**

**auto** ошибка **=** **new** ОшибкаЛексера**(**положение**,** сооб**);**

ошибки **~=** ошибка**;**

**if** **(**диаг **!is** **null)**

диаг **~=** ошибка**;**

**}**

/// Scans the whole source текст until КФ is encountered.

проц сканируйВсе**()**

**{**

**while** **(**следщСема**()** **!=** TOK**.**КФ**)**

**{}**

**}**

/// Возвращает first сема of the source текст.

/// Этот can be the КФ сема.

/// Structure: ГОЛОВА -> Новстр -> First Сема

Сема**\*** перваяСема**()**

**{**

**return** **this.**глава**.**следщ**.**следщ**;**

**}**

/// Returns да if ткт is a valid D identifier.

**static** бул строкаИдентификатора\_ли**(**ткст ткт**)**

**{**

**if** **(**ткт**.**length **==** 0 **||** цифра\_ли**(**ткт**[**0**]))**

**return** нет**;**

т\_мера idx**;**

**do**

**{**

**auto** c **=** drc**.**Unicode**.**раскодируй**(**ткт**,** idx**);**

**if** **(**c **==** СИМ\_ОШИБКИ **||** **!(**идент\_ли**(**c**)** **||** **!**аски\_ли**(**c**)** **&&** униАльфа\_ли**(**c**)))**

**return** нет**;**

**}** **while** **(**idx **<** ткт**.**length**)**

**return** да**;**

**}**

/// Returns да if ткт is a keyword or

/// a special сема (\_\_FILE\_\_, \_\_LINE\_\_ etc.)

**static** бул резервныйИдентификатор\_ли**(**ткст ткт**)**

**{**

**if** **(**ткт**.**length **==** 0**)**

**return** нет**;**

**auto** ид **=** ТаблицаИд**.**вСтатической**(**ткт**);**

**if** **(**ид **is** **null** **||** ид**.**вид **==** TOK**.**Идентификатор**)**

**return** нет**;** // ткт is not in the таблица or a normal identifier.

**return** да**;**

**}**

/// Возвращает да, если из\_ a valid identifier and if it's not reserved.

**static** бул действитНерезИдентификатор\_ли**(**ткст ткт**)**

**{**

**return** строкаИдентификатора\_ли**(**ткт**)** **&&** **!**резервныйИдентификатор\_ли**(**ткт**);**

**}**

/// Returns да if the current символ в be decoded is

/// a Unicode alpha символ.

///

/// The current pointer 'p' is установи в the last trailbyte if да is returned.

бул юАльфа\_ли**()**

**{**

**assert(!**аски\_ли**(\***p**),** "check for ASCII сим before calling раскодируйЮ8()."**);**

сим**\*** p **=** **this.**p**;**

дим d **=** **\***p**;**

**++**p**;** // Move в second байт.

// Ошибка if second байт is not a trail байт.

**if** **(!**ведомыйБайт\_ли**(\***p**))**

**return** нет**;**

// Check for overlong sequences.

**switch** **(**d**)**

**{**

**case** 0xE0**,** 0xF0**,** 0xF8**,** 0xFC**:**

**if** **((\***p **&** d**)** **==** 0x80**)**

**return** нет**;**

**default:**

**if** **((**d **&** 0xFE**)** **==** 0xC0**)** // 1100000x

**return** нет**;**

**}**

**const** ткст проверьСледующийБайт **=** "if (!ведомыйБайт\_ли(\*++p))"

" return нет;"**;**

**const** ткст добавьШестьБит **=** "d = (d << 6) | \*p & 0b0011\_1111;"**;**

// Decode

**if** **((**d **&** 0b1110\_0000**)** **==** 0b1100\_0000**)**

**{**

d **&=** 0b0001\_1111**;**

**mixin(**добавьШестьБит**);**

**}**

**else** **if** **((**d **&** 0b1111\_0000**)** **==** 0b1110\_0000**)**

**{**

d **&=** 0b0000\_1111**;**

**mixin(**добавьШестьБит **~**

проверьСледующийБайт **~** добавьШестьБит**);**

**}**

**else** **if** **((**d **&** 0b1111\_1000**)** **==** 0b1111\_0000**)**

**{**

d **&=** 0b0000\_0111**;**

**mixin(**добавьШестьБит **~**

проверьСледующийБайт **~** добавьШестьБит **~**

проверьСледующийБайт **~** добавьШестьБит**);**

**}**

**else**

**return** нет**;**

**assert(**ведомыйБайт\_ли**(\***p**));**

**if** **(!**верноСимвол\_ли**(**d**)** **||** **!**униАльфа\_ли**(**d**))**

**return** нет**;**

// Only advance pointer if this is a Unicode alpha символ.

**this.**p **=** p**;**

**return** да**;**

**}**

/// Decodes the следщ UTF-8 sequence.

дим раскодируйЮ8**()**

**{**

**assert(!**аски\_ли**(\***p**),** "check for ASCII char before calling раскодируйЮ8()."**);**

сим**\*** p **=** **this.**p**;**

дим d **=** **\***p**;**

**++**p**;** // Move в second байт.

// Ошибка if second байт is not a trail байт.

**if** **(!**ведомыйБайт\_ли**(\***p**))**

**goto** Lerr2**;**

// Check for overlong sequences.

**switch** **(**d**)**

**{**

**case** 0xE0**,** // 11100000 100xxxxx

0xF0**,** // 11110000 1000xxxx

0xF8**,** // 11111000 10000xxx

0xFC**:** // 11111100 100000xx

**if** **((\***p **&** d**)** **==** 0x80**)**

**goto** Lerr**;**

**default:**

**if** **((**d **&** 0xFE**)** **==** 0xC0**)** // 1100000x

**goto** Lerr**;**

**}**

**const** ткст проверьСледующийБайт **=** "if (!ведомыйБайт\_ли(\*++p))"

" goto Lerr2;"**;**

**const** ткст добавьШестьБит **=** "d = (d << 6) | \*p & 0b0011\_1111;"**;**

// Decode

**if** **((**d **&** 0b1110\_0000**)** **==** 0b1100\_0000**)**

**{** // 110xxxxx 10xxxxxx

d **&=** 0b0001\_1111**;**

**mixin(**добавьШестьБит**);**

**}**

**else** **if** **((**d **&** 0b1111\_0000**)** **==** 0b1110\_0000**)**

**{** // 1110xxxx 10xxxxxx 10xxxxxx

d **&=** 0b0000\_1111**;**

**mixin(**добавьШестьБит **~**

проверьСледующийБайт **~** добавьШестьБит**);**

**}**

**else** **if** **((**d **&** 0b1111\_1000**)** **==** 0b1111\_0000**)**

**{** // 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

d **&=** 0b0000\_0111**;**

**mixin(**добавьШестьБит **~**

проверьСледующийБайт **~** добавьШестьБит **~**

проверьСледующийБайт **~** добавьШестьБит**);**

**}**

**else**

// 5 and 6 байт UTF-8 sequences are not allowed yet.

// 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

// 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

**goto** Lerr**;**

**assert(**ведомыйБайт\_ли**(\***p**));**

**if** **(!**верноСимвол\_ли**(**d**))**

**{**

Lerr**:**

// Three cases:

// \*) the UTF-8 sequence was successfully decoded but the resulting

// символ is invalid.

// p points в last trail байт in the sequence.

// \*) the UTF-8 sequence is overlong.

// p points в second байт in the sequence.

// \*) the UTF-8 sequence has more than 4 bytes or starts with

// a trail байт.

// p points в second байт in the sequence.

**assert(**ведомыйБайт\_ли**(\***p**));**

// Move в следщ ASCII символ or lead байт of a UTF-8 sequence.

**while** **(**p **<** **(**конец**-**1**)** **&&** ведомыйБайт\_ли**(\***p**))**

**++**p**;**

**--**p**;**

**assert(!**ведомыйБайт\_ли**(**p**[**1**]));**

Lerr2**:**

d **=** СИМ\_ЗАМЕНЫ**;**

ошибка**(this.**p**,** ИДС**.**НедействительнаяПоследовательностьУТФ8**,** formatBytes**(this.**p**,** p**));**

**}**

**this.**p **=** p**;**

**return** d**;**

**}**

/// Encodes the символ d and appends it в ткт.

**static** проц encodeUTF8**(**ref ткст ткт**,** дим d**)**

**{**

**assert(!**аски\_ли**(**d**),** "check for ASCII сим before calling encodeUTF8()."**);**

**assert(**верноСимвол\_ли**(**d**),** "check if символ is valid before calling encodeUTF8()."**);**

сим**[**6**]** b **=** **void;**

**if** **(**d **<** 0x800**)**

**{**

b**[**0**]** **=** 0xC0 **|** **(**d **>>** 6**);**

b**[**1**]** **=** 0x80 **|** **(**d **&** 0x3F**);**

ткт **~=** b**[**0**..**2**];**

**}**

**else** **if** **(**d **<** 0x10000**)**

**{**

b**[**0**]** **=** 0xE0 **|** **(**d **>>** 12**);**

b**[**1**]** **=** 0x80 **|** **((**d **>>** 6**)** **&** 0x3F**);**

b**[**2**]** **=** 0x80 **|** **(**d **&** 0x3F**);**

ткт **~=** b**[**0**..**3**];**

**}**

**else** **if** **(**d **<** 0x200000**)**

**{**

b**[**0**]** **=** 0xF0 **|** **(**d **>>** 18**);**

b**[**1**]** **=** 0x80 **|** **((**d **>>** 12**)** **&** 0x3F**);**

b**[**2**]** **=** 0x80 **|** **((**d **>>** 6**)** **&** 0x3F**);**

b**[**3**]** **=** 0x80 **|** **(**d **&** 0x3F**);**

ткт **~=** b**[**0**..**4**];**

**}**

/+ // There are no 5 and 6 байт UTF-8 sequences yet.

else if (d < 0x4000000)

{

b[0] = 0xF8 | (d >> 24);

b[1] = 0x80 | ((d >> 18) & 0x3F);

b[2] = 0x80 | ((d >> 12) & 0x3F);

b[3] = 0x80 | ((d >> 6) & 0x3F);

b[4] = 0x80 | (d & 0x3F);

ткт ~= b[0..5];

}

else if (d < 0x80000000)

{

b[0] = 0xFC | (d >> 30);

b[1] = 0x80 | ((d >> 24) & 0x3F);

b[2] = 0x80 | ((d >> 18) & 0x3F);

b[3] = 0x80 | ((d >> 12) & 0x3F);

b[4] = 0x80 | ((d >> 6) & 0x3F);

b[5] = 0x80 | (d & 0x3F);

ткт ~= b[0..6];

}

+/

**else**

**assert(**0**);**

**}**

/// Formats the bytes between старт and конец.

/// Возвращает: в.g.: abc -> \x61\x62\x63

**static** ткст formatBytes**(**сим**\*** старт**,** сим**\*** конец**)**

**{**

**auto** тктLen **=** конец**-**старт**;**

**const** formatLen **=** `\xXX`**.**length**;**

ткст результат **=** **new** сим**[**тктLen**\***formatLen**];** // Reserve space.

результат**.**length **=** 0**;**

**foreach** **(**c**;** **cast(**ббайт**[])**старт**[**0**..**тктLen**])**

результат **~=** Формат**(**"\\x{:X}"**,** c**);**

**return** результат**;**

**}**

/// Searches for an invalid UTF-8 sequence in ткт.

/// Возвращает: a formatted ткст of the invalid sequence (в.g. \xC0\x80).

**static** ткст найдиНедействительнуюПоследовательностьУТФ8**(**ткст ткт**)**

**{**

сим**\*** p **=** ткт**.**ptr**,** конец **=** p **+** ткт**.**length**;**

**while** **(**p **<** конец**)**

**{**

**if** **(**раскодируй**(**p**,** конец**)** **==** СИМ\_ОШИБКИ**)**

**{**

**auto** начало **=** p**;**

// Skip trail-bytes.

**while** **(++**p **<** конец **&&** ведомыйБайт\_ли**(\***p**))**

**{}**

**return** Лексер**.**formatBytes**(**начало**,** p**);**

**}**

**}**

**assert(**p **==** конец**);**

**return** ""**;**

**}**

**}**

/// Tests the лексер with a список of семы.

**unittest**

**{**

выдай**(**"Тестируем Лексер.\n"**);**

**struct** Пара

**{**

ткст текстТокена**;**

TOK вид**;**

**}**

**static** Пара**[]** пары **=** **[**

**{**"#!äöüß"**,** TOK**.**Шебанг**},** **{**"\n"**,** TOK**.**Новстр**},**

**{**"//çay"**,** TOK**.**Комментарий**},** **{**"\n"**,** TOK**.**Новстр**},**

**{**"&"**,** TOK**.**ИБинарное**},**

**{**"/\*çağ\*/"**,** TOK**.**Комментарий**},** **{**"&&"**,** TOK**.**ИЛогическое**},**

**{**"/+çak+/"**,** TOK**.**Комментарий**},** **{**"&="**,** TOK**.**ИПрисвой**},**

**{**">"**,** TOK**.**Больше**},** **{**"+"**,** TOK**.**Плюс**},**

**{**">="**,** TOK**.**БольшеРавно**},** **{**"++"**,** TOK**.**ПлюсПлюс**},**

**{**">>"**,** TOK**.**ПСдвиг**},** **{**"+="**,** TOK**.**ПлюсПрисвой**},**

**{**">>="**,** TOK**.**ПСдвигПрисвой**},** **{**"-"**,** TOK**.**Минус**},**

**{**">>>"**,** TOK**.**URShift**},** **{**"--"**,** TOK**.**МинусМинус**},**

**{**">>>="**,** TOK**.**URShiftAssign**},** **{**"-="**,** TOK**.**МинусПрисвой**},**

**{**"<"**,** TOK**.**Меньше**},** **{**"="**,** TOK**.**Присвоить**},**

**{**"<="**,** TOK**.**МеньшеРавно**},** **{**"=="**,** TOK**.**Равно**},**

**{**"<>"**,** TOK**.**LorG**},** **{**"~"**,** TOK**.**Тильда**},**

**{**"<>="**,** TOK**.**LorEorG**},** **{**"~="**,** TOK**.**CatAssign**},**

**{**"<<"**,** TOK**.**ЛСдвиг**},** **{**"\*"**,** TOK**.**Умножь**},**

**{**"<<="**,** TOK**.**ЛСдвигПрисвой**},** **{**"\*="**,** TOK**.**УмножьПрисвой**},**

**{**"!"**,** TOK**.**Не**},** **{**"/"**,** TOK**.**Деление**},**

**{**"!="**,** TOK**.**НеРавно**},** **{**"/="**,** TOK**.**ДелениеПрисвой**},**

**{**"!<"**,** TOK**.**UorGorE**},** **{**"^"**,** TOK**.**ИИли**},**

**{**"!>"**,** TOK**.**UorLorE**},** **{**"^="**,** TOK**.**ИИлиПрисвой**},**

**{**"!<="**,** TOK**.**UorG**},** **{**"%"**,** TOK**.**Модуль**},**

**{**"!>="**,** TOK**.**UorL**},** **{**"%="**,** TOK**.**МодульПрисвой**},**

**{**"!<>"**,** TOK**.**UorE**},** **{**"("**,** TOK**.**ЛСкобка**},**

**{**"!<>="**,** TOK**.**Unordered**},** **{**")"**,** TOK**.**ПСкобка**},**

**{**"."**,** TOK**.**Точка**},** **{**"["**,** TOK**.**ЛКвСкобка**},**

**{**".."**,** TOK**.**Срез**},** **{**"]"**,** TOK**.**ПКвСкобка**},**

**{**"..."**,** TOK**.**Эллипсис**},** **{**"{"**,** TOK**.**ЛФСкобка**},**

**{**"|"**,** TOK**.**ИлиБинарное**},** **{**"}"**,** TOK**.**ПФСкобка**},**

**{**"||"**,** TOK**.**ИлиЛогическое**},** **{**":"**,** TOK**.**Двоеточие**},**

**{**"|="**,** TOK**.**ИлиПрисвой**},** **{**";"**,** TOK**.**ТочкаЗапятая**},**

**{**"?"**,** TOK**.**Вопрос**},** **{**","**,** TOK**.**Запятая**},**

**{**"$"**,** TOK**.**Доллар**},** **{**"cam"**,** TOK**.**Идентификатор**},**

**{**"çay"**,** TOK**.**Идентификатор**},** **{**".0"**,** TOK**.**Плав64**},**

**{**"0"**,** TOK**.**Цел32**},** **{**"\n"**,** TOK**.**Новстр**},**

**{**"\r"**,** TOK**.**Новстр**},** **{**"\r\n"**,** TOK**.**Новстр**},**

**{**"\u2028"**,** TOK**.**Новстр**},** **{**"\u2029"**,** TOK**.**Новстр**}**

**];**

ткст ист**;**

// Join all сема тексты into a single ткст.

**foreach** **(**i**,** пара**;** пары**)**

**if** **(**пара**.**вид **==** TOK**.**Комментарий **&&** пара**.**текстТокена**[**1**]** **==** '/' **||** // Line comment.

пара**.**вид **==** TOK**.**Шебанг**)**

**{**

**assert(**пары**[**i**+**1**].**вид **==** TOK**.**Новстр**);** // Must be followed by a нс.

ист **~=** пара**.**текстТокена**;**

**}**

**else**

ист **~=** пара**.**текстТокена **~** " "**;**

// Lex the constructed source текст.

**auto** lx **=** **new** Лексер**(new** ИсходныйТекст**(**""**,** ист**));**

lx**.**сканируйВсе**();**

**auto** сема **=** lx**.**перваяСема**();**

**for** **(**бцел i**;** i **<** пары**.**length **&&** сема**.**вид **!=** TOK**.**КФ**;**

**++**i**,** **(**сема **=** сема**.**следщ**))**

**if** **(**сема**.**исхТекст **!=** пары**[**i**].**текстТокена**)**

**assert(**0**,** Формат**(**"Найдено '{0}' , но ожидалось '{1}'"**,**

сема**.**исхТекст**,** пары**[**i**].**текстТокена**));**

**}**

/// Tests the Лексер's возьми() method.

**unittest**

**{**

выдай**(**"Тестируем метод Лексер.возьми()\n"**);**

**auto** исходныйТекст **=** **new** ИсходныйТекст**(**""**,** "unittest { }"**);**

**auto** lx **=** **new** Лексер**(**исходныйТекст**,** **null);**

**auto** следщ **=** lx**.**глава**;**

lx**.**возьми**(**следщ**);**

**assert(**следщ**.**вид **==** TOK**.**Новстр**);**

lx**.**возьми**(**следщ**);**

**assert(**следщ**.**вид **==** TOK**.**Юниттест**);**

lx**.**возьми**(**следщ**);**

**assert(**следщ**.**вид **==** TOK**.**ЛФСкобка**);**

lx**.**возьми**(**следщ**);**

**assert(**следщ**.**вид **==** TOK**.**ПФСкобка**);**

lx**.**возьми**(**следщ**);**

**assert(**следщ**.**вид **==** TOK**.**КФ**);**

lx **=** **new** Лексер**(new** ИсходныйТекст**(**""**,** ""**));**

следщ **=** lx**.**глава**;**

lx**.**возьми**(**следщ**);**

**assert(**следщ**.**вид **==** TOK**.**Новстр**);**

lx**.**возьми**(**следщ**);**

**assert(**следщ**.**вид **==** TOK**.**КФ**);**

**}**

**unittest**

**{**

// Numbers unittest

// 0L 0ULi 0\_L 0\_UL 0x0U 0x0p2 0\_Fi 0\_e2 0\_F 0\_i

// 0u 0U 0uL 0UL 0L 0LU 0Lu

// 0Li 0f 0F 0fi 0Fi 0i

// 0b\_1\_LU 0b1000u

// 0x232Lu

**}**

**module drc.lexer.Token;**

**import** drc**.**lexer**.**Identifier**,**

drc**.**lexer**.**Funcs**;**

**import** drc**.**Location**;**

**import** cidrus **:** malloc**,** free**;**

**import** exception**;**

**import** common**;**

**public** **import** drc**.**lexer**.**TokensEnum**;**

/// A Сема - из\_ цепочка символов, формируемая лексическим анализатором.

**struct** **Сема**

**{** /// Флаги, устанавливаемые Лексером.

**enum** **Флаги** **:** бкрат

**{**

Нет**,**

Пробельный **=** 1**,** /// Знаки с этим флагом игнорируются Парсером.

**}**

TOK вид**;** /// Вид семы.

Флаги флаги**;** /// Флаги семы.

/// Указатели на следующую и предыдущую семы (дважды линкованный список.)

Сема**\*** следщ**,** предш**;**

/// Начало пробельных символов перед семой. Нуль, если их нет.

/// TODO: remove в save space; can be replaced by 'предш.конец'.

сим**\*** пп**;**

сим**\*** старт**;** /// Указывает на первый символ семы.

сим**\*** конец**;** /// Points one символ past the конец of the сема.

/// Данные, ассоциированные с данной семой.

/// TODO: move данные structures out; use only pointers here в keep Сема.sizeof small.

**union**

**{**

/// При нс семы.

ДанныеНовСтр нс**;**

/// При #line семы.

**struct**

**{**

Сема**\*** tokLineNum**;** /// #line число

Сема**\*** tokLineFilespec**;** /// #line число filespec

**}**

/// The значение of a ткст сема.

**struct**

**{**

ткст ткт**;** /// Zero-terminated ткст. (The zero is included in the length.)

сим pf**;** /// Postfix 'c', 'w', 'd' or 0 for none.

**version(**D2**)**

Сема**\*** tok\_ткт**;** /++ Points в the contents of a сема ткст stored as a

doubly linked список. The last сема is always '}' or

КФ in case конец of source текст is "q{" КФ.

+/

**}**

Идентификатор**\*** идент**;** /// При keywords and identifiers.

дим дим\_**;** /// A символ значение.

дол дол\_**;** /// A дол integer значение.

бдол бдол\_**;** /// An unsigned дол integer значение.

цел цел\_**;** /// An integer значение.

бцел бцел\_**;** /// An unsigned integer значение.

плав плав\_**;** /// A плав значение.

дво дво\_**;** /// A дво значение.

реал реал\_**;** /// A реал значение.

**}**

/// Возвращает текст of the сема.

ткст исхТекст**()**

**{**

**assert(**старт **&&** конец**);**

**return** старт**[**0 **..** конец **-** старт**];**

**}**

/// Возвращает preceding whitespace of the сема.

ткст пробСимволы**()**

**{**

**assert(**пп **&&** старт**);**

**return** пп**[**0 **..** старт **-** пп**];**

**}**

/// Finds the следщ non-whitespace сема.

/// Возвращает: 'эту' сему, если предыдущая является TOK.ГОЛОВА или null.

Сема**\*** следщНепроб**()**

**out(**сема**)**

**{**

**assert(**сема **!is** **null);**

**}**

**body**

**{**

**auto** сема **=** следщ**;**

**while** **(**сема **!is** **null** **&&** сема**.**пробел\_ли**)**

сема **=** сема**.**следщ**;**

**if** **(**сема **is** **null** **||** сема**.**вид **==** TOK**.**КФ**)**

**return** **this;**

**return** сема**;**

**}**

/// Находит предшествующую непробельную сему.

/// Возвращает: 'эту' сему, если предыдущая является TOK.ГОЛОВА или null.

Сема**\*** предшНепроб**()**

**out(**сема**)**

**{**

**assert(**сема **!is** **null);**

**}**

**body**

**{**

**auto** сема **=** предш**;**

**while** **(**сема **!is** **null** **&&** сема**.**пробел\_ли**)**

сема **=** сема**.**предш**;**

**if** **(**сема **is** **null** **||** сема**.**вид **==** TOK**.**ГОЛОВА**)**

**return** **this;**

**return** сема**;**

**}**

/// Возвращает текстовое определение вида данной семы.

**static** ткст вТкст**(**TOK вид**)**

**{**

**return** семаВТкст**[**вид**];**

**}**

/// Adds Флаги.Пробельный в this.флаги.

проц установиФлагПробельные**()**

**{**

**this.**флаги **|=** Флаги**.**Пробельный**;**

**}**

/// Возвращает да, если из\_ сема, внутри которой могут быть символы новой строки.

///

/// These can be block and nested comments and any ткст literal

/// except for escape ткст literals.

бул многострок\_ли**()**

**{**

**return** вид **==** TOK**.**Ткст **&&** старт**[**0**]** **!=** '\\' **||**

вид **==** TOK**.**Комментарий **&&** старт**[**1**]** **!=** '/'**;**

**}**

/// Возвращает да, если из\_ сема-ключевое слово.

бул кслово\_ли**()**

**{**

**return** НачалоКС **<=** вид **&&** вид **<=** КонецКС**;**

**}**

/// Возвращает да, если из\_ сема интегрального типа.

бул интегральныйТип\_ли**()**

**{**

**return** НачалоИнтегральногоТипа **<=** вид **&&** вид **<=** КонецИнтегральногоТипа**;**

**}**

/// Возвращает да, если из\_ сема пробела.

бул пробел\_ли**()**

**{**

**return** **!!(**флаги **&** Флаги**.**Пробельный**);**

**}**

/// Возвращает да, если из\_ a special сема.

бул спецСема\_ли**()**

**{**

**return** НачалоСпецСем **<=** вид **&&** вид **<=** КонецСпецСем**;**

**}**

**version(**D2**)**

**{**

/// Возвращает да, если из\_ a сема ткст literal.

бул семаСтроковогоЛитерала\_ли**()**

**{**

**return** вид **==** TOK**.**Ткст **&&** tok\_ткт **!is** **null;**

**}**

**}**

/// Returns да if this сема starts a ДефиницияДекларации.

бул началоДефДекл\_ли**()**

**{**

**return** семаНачалаДеклДеф\_ли**(**вид**);**

**}**

/// Returns да if this сема starts a Инструкция.

бул началоИнстр\_ли**()**

**{**

**return** семаНачалаИнстр\_ли**(**вид**);**

**}**

/// Returns да if this сема starts an ИнструкцияАсм.

бул началоАсмИнстр\_ли**()**

**{**

**return** семаНачалаАсмИнстр\_ли**(**вид**);**

**}**

цел opEquals**(**TOK kind2**)**

**{**

**return** вид **==** kind2**;**

**}**

цел opCmp**(**Сема**\*** пв**)**

**{**

**return** старт **<** пв**.**старт**;**

**}**

/// Возвращает Положение of this сема.

Положение дайПоложение**(**бул реальноеПоложение**)()**

**{**

**auto** search\_t **=** **this.**предш**;**

// Find предшious нс сема.

**while** **(**search\_t**.**вид **!=** TOK**.**Новстр**)**

search\_t **=** search\_t**.**предш**;**

**static** **if** **(**реальноеПоложение**)**

**{**

**auto** путьКФайлу **=** search\_t**.**нс**.**путиКФайлам**.**исхПуть**;**

**auto** номСтр **=** search\_t**.**нс**.**oriLineNum**;**

**}**

**else**

**{**

**auto** путьКФайлу **=** search\_t**.**нс**.**путиКФайлам**.**устПуть**;**

**auto** номСтр **=** search\_t**.**нс**.**oriLineNum **-** search\_t**.**нс**.**setLineNum**;**

**}**

**auto** началоСтроки **=** search\_t**.**конец**;**

// Determine actual line начало and line число.

**while** **(**1**)**

**{**

search\_t **=** search\_t**.**следщ**;**

**if** **(**search\_t **==** **this)**

**break;**

// Multiline семы must be rescanned for newlines.

**if** **(**search\_t**.**многострок\_ли**)**

**{**

**auto** p **=** search\_t**.**старт**,** конец **=** search\_t**.**конец**;**

**while** **(**p **!=** конец**)**

**if** **(**сканируйНовСтр**(**p**))**

**{**

началоСтроки **=** p**;**

**++**номСтр**;**

**}**

**else**

**++**p**;**

**}**

**}**

**return** **new** Положение**(**путьКФайлу**,** номСтр**,** началоСтроки**,** **this.**старт**);**

**}**

**alias** дайПоложение**!(**да**)** дайРеальноеПоложение**;**

**alias** дайПоложение**!(**нет**)** дайПоложениеОшибки**;**

бцел lineCount**()**

**{**

бцел счёт **=** 1**;**

**if** **(this.**многострок\_ли**)**

**{**

**auto** p **=** **this.**старт**,** конец **=** **this.**конец**;**

**while** **(**p **!=** конец**)**

**{**

**if** **(**сканируйНовСтр**(**p**)** **==** '\n'**)**

**++**счёт**;**

**else**

**++**p**;**

**}**

**}**

**return** счёт**;**

**}**

/// Итог the source текст enclosed by the левый and правый сема.

**static** ткст textSpan**(**Сема**\*** левый**,** Сема**\*** правый**)**

**{**

**assert(**левый**.**конец **<=** правый**.**старт **||** левый **is** правый **);**

**return** левый**.**старт**[**0 **..** правый**.**конец **-** левый**.**старт**];**

**}**

/// Uses malloc() в allocate memory for a сема.

**new(**т\_мера размер**)**

**{**

ук p **=** malloc**(**размер**);**

**if** **(**p **is** **null)**

**throw** **new** OutOfMemoryException**(**\_\_FILE\_\_**,** \_\_LINE\_\_**);**

// TODO: Сема.иниц should be all zeros.

// Maybe use calloc() в avoid this line?

**\*cast(**Сема**\*)**p **=** Сема**.**init**;**

**return** p**;**

**}**

/// Deletes a сема using free().

**delete(**ук p**)**

**{**

**auto** сема **=** **cast(**Сема**\*)**p**;**

**if** **(**сема**)**

**{**

**if(**сема**.**вид **==** TOK**.**HashLine**)**

сема**.**destructHashLineToken**();**

**else**

**{**

**version(**D2**)**

**if** **(**сема**.**семаСтроковогоЛитерала\_ли**)**

сема**.**destructTokenStringLiteral**();**

**}**

**}**

free**(**p**);**

**}**

проц destructHashLineToken**()**

**{**

**assert(**вид **==** TOK**.**HashLine**);**

**delete** tokLineNum**;**

**delete** tokLineFilespec**;**

**}**

**version(**D2**)**

**{**

проц destructTokenStringLiteral**()**

**{**

**assert(**вид **==** TOK**.**Ткст**);**

**assert(**старт **&&** **\***старт **==** 'q' **&&** старт**[**1**]** **==** '{'**);**

**assert(**tok\_ткт **!is** **null);**

**auto** tok\_it **=** tok\_ткт**;**

**auto** tok\_del **=** tok\_ткт**;**

**while** **(**tok\_it **&&** tok\_it**.**вид **!=** TOK**.**КФ**)**

**{**

tok\_it **=** tok\_it**.**следщ**;**

**assert(**tok\_del **&&** tok\_del**.**вид **!=** TOK**.**КФ**);**

**delete** tok\_del**;**

tok\_del **=** tok\_it**;**

**}**

**}**

**}**

**}**

/// Data associated with нс семы.

**struct** ДанныеНовСтр

**{**

**struct** ФПути

**{**

ткст исхПуть**;** /// Original путь в the source текст.

ткст устПуть**;** /// Path установи by #line.

**}**

ФПути**\*** путиКФайлам**;**

бцел oriLineNum**;** /// Actual line число in the source текст.

бцел setLineNum**;** /// Delta line число установи by #line.

**}**

/// Returns да if this сема starts a ДефиницияДекларации.

бул семаНачалаДеклДеф\_ли**(**TOK лекс**)**

**{**

**switch** **(**лекс**)**

**{**

**alias** TOK T**;**

**case** T**.**Расклад**,** T**.**Прагма**,** T**.**Экспорт**,** T**.**Приватный**,** T**.**Пакет**,** T**.**Защищённый**,**

T**.**Публичный**,** T**.**Экстерн**,** T**.**Устаревший**,** T**.**Перепись**,** T**.**Абстрактный**,**

T**.**Синхронизованный**,** T**.**Статический**,** T**.**Окончательный**,** T**.**Конст**,** T**.**Инвариант/\*D 2.0\*/**,**

T**.**Авто**,** T**.**Масштаб**,** T**.**Алиас**,** T**.**Типдеф**,** T**.**Импорт**,** T**.**Перечень**,** T**.**Класс**,**

T**.**Интерфейс**,** T**.**Структура**,** T**.**Союз**,** T**.**Этот**,** T**.**Тильда**,** T**.**Юниттест**,** T**.**Отладка**,**

T**.**Версия**,** T**.**Шаблон**,** T**.**Нов**,** T**.**Удалить**,** T**.**Смесь**,** T**.**ТочкаЗапятая**,**

T**.**Идентификатор**,** T**.**Точка**,** T**.**Типа**:**

**return** да**;**

**default:**

**if** **(**НачалоИнтегральногоТипа **<=** лекс **&&** лекс **<=** КонецИнтегральногоТипа**)**

**return** да**;**

**}**

**return** нет**;**

**}**

/// Returns да if this сема starts a Инструкция.

бул семаНачалаИнстр\_ли**(**TOK лекс**)**

**{**

**switch** **(**лекс**)**

**{**

**alias** TOK T**;**

**case** T**.**Расклад**,** T**.**Экстерн**,** T**.**Окончательный**,** T**.**Конст**,** T**.**Авто**,** T**.**Идентификатор**,** T**.**Точка**,**

T**.**Типа**,** T**.**Если**,** T**.**Пока**,** T**.**Делай**,** T**.**При**,** T**.**Длявсех**,** T**.**Длявсех\_реверс**,**

T**.**Щит**,** T**.**Реле**,** T**.**Дефолт**,** T**.**Далее**,** T**.**Всё**,** T**.**Итог**,** T**.**Переход**,**

T**.**Для**,** T**.**Синхронизованный**,** T**.**Пробуй**,** T**.**Брось**,** T**.**Масштаб**,** T**.**Волатайл**,** T**.**Асм**,**

T**.**Прагма**,** T**.**Смесь**,** T**.**Статический**,** T**.**Отладка**,** T**.**Версия**,** T**.**Алиас**,** T**.**ТочкаЗапятая**,**

T**.**Перечень**,** T**.**Класс**,** T**.**Интерфейс**,** T**.**Структура**,** T**.**Союз**,** T**.**ЛФСкобка**,** T**.**Типдеф**,**

T**.**Этот**,** T**.**Супер**,** T**.**Нуль**,** T**.**Истина**,** T**.**Ложь**,** T**.**Цел32**,** T**.**Цел64**,** T**.**Бцел32**,**

T**.**Бцел64**,** T**.**Плав32**,** T**.**Плав64**,** T**.**Плав80**,** T**.**Мнимое32**,**

T**.**Мнимое64**,** T**.**Мнимое80**,** T**.**СимЛитерал**,** T**.**Ткст**,** T**.**ЛКвСкобка**,**

T**.**Функция**,** T**.**Делегат**,** T**.**Подтвердить**,** T**.**Импорт**,** T**.**Идтипа**,** T**.**Является**,** T**.**ЛСкобка**,**

T**.**Трэтс/\*D2.0\*/**,** T**.**ИБинарное**,** T**.**ПлюсПлюс**,** T**.**МинусМинус**,** T**.**Умножь**,**

T**.**Минус**,** T**.**Плюс**,** T**.**Не**,** T**.**Тильда**,** T**.**Нов**,** T**.**Удалить**,** T**.**Каст**:**

**return** да**;**

**default:**

**if** **(**НачалоИнтегральногоТипа **<=** лекс **&&** лекс **<=** КонецИнтегральногоТипа **||**

НачалоСпецСем **<=** лекс **&&** лекс **<=** КонецСпецСем**)**

**return** да**;**

**}**

**return** нет**;**

**}**

/// Returns да if this сема starts an ИнструкцияАсм.

бул семаНачалаАсмИнстр\_ли**(**TOK лекс**)**

**{**

**switch(**лекс**)**

**{**

**alias** TOK T**;**

**case** T**.**Вхо**,** T**.**Цел**,** T**.**Вых**,** T**.**Идентификатор**,** T**.**Расклад**,** T**.**ТочкаЗапятая**:**

**return** да**;**

**default:**

**}**

**return** нет**;**

**}**

**module drc.lexer.TokensEnum;**

**import** common**;**

/// Перечисление типов сем.

**enum** **TOK** **:** бкрат

**{**

Неверно**,**

Нелегал**,**

Комментарий**,**

Шебанг**,**

HashLine**,**

Filespec**,**

Новстр**,**

Пусто**,**

Идентификатор**,**

Ткст**,**

СимЛитерал**,**

// Специальные семы

ФАЙЛ**,**

СТРОКА**,**

ДАТА**,**

ВРЕМЯ**,**

ШТАМПВРЕМЕНИ**,**

ПОСТАВЩИК**,**

ВЕРСИЯ**,**

// Числовые литералы

Цел32**,** Цел64**,** Бцел32**,** Бцел64**,**

// Сканер чисел с плавающей точкой рассчитывает на такой расклад. (ПлавXY + 3 == МнимоеXY)

Плав32**,** Плав64**,** Плав80**,**

Мнимое32**,** Мнимое64**,** Мнимое80**,**

// Скобки

ЛСкобка**,**

ПСкобка**,**

ЛКвСкобка**,**

ПКвСкобка**,**

ЛФСкобка**,**

ПФСкобка**,**

Точка**,** Срез**,** Эллипсис**,**

// Операты над числами с плавающей точкой

Unordered**,**

UorE**,**

UorG**,**

UorGorE**,**

UorL**,**

UorLorE**,**

LorEorG**,**

LorG**,**

// Нормальные операторы

Присвоить**,** Равно**,** НеРавно**,** Не**,**

МеньшеРавно**,** Меньше**,**

БольшеРавно**,** Больше**,**

ЛСдвигПрисвой**,** ЛСдвиг**,**

ПСдвигПрисвой**,**ПСдвиг**,**

URShiftAssign**,** URShift**,**

ИлиПрисвой**,** ИлиЛогическое**,** ИлиБинарное**,**

ИПрисвой**,** ИЛогическое**,** ИБинарное**,**

ПлюсПрисвой**,** ПлюсПлюс**,** Плюс**,**

МинусПрисвой**,** МинусМинус**,** Минус**,**

ДелениеПрисвой**,** Деление**,**

УмножьПрисвой**,** Умножь**,**

МодульПрисвой**,** Мод**,**

ИИлиПрисвой**,** ИИли**,**

CatAssign**,**

Тильда**,**

Двоеточие**,**

ТочкаЗапятая**,**

Вопрос**,**

Запятая**,**

Доллар**,**

/\* Keywords:

NB.: Сема.кслово\_ли() depends on this список being contiguous.

\*/

Абстрактный**,** Алиас**,** Расклад**,** Асм**,** Подтвердить**,** Авто**,** Тело**,**

Всё**,** Реле**,** Каст**,** Кэтч**,**

Класс**,** Конст**,** Далее**,**

Отладка**,** Дефолт**,** Делегат**,** Удалить**,** Устаревший**,** Делай**,**

Иначе**,** Перечень**,** Экспорт**,** Экстерн**,** Ложь**,** Окончательный**,**

Finally**,** При**,** Длявсех**,** Длявсех\_реверс**,** Функция**,** Переход**,**

Если**,** Импорт**,** Вхо**,** Вховых**,**

Интерфейс**,** Инвариант**,** Является**,** Отложенный**,** Макрос/+D2.0+/**,**

Смесь**,** Модуль**,** Нов**,** Nothrow/+D2.0+/**,** Нуль**,** Вых**,** Перепись**,** Пакет**,**

Прагма**,** Приватный**,** Защищённый**,** Публичный**,** Pure/+D2.0+/**,** Реф**,** Итог**,**

Масштаб**,** Статический**,** Структура**,** Супер**,** Щит**,** Синхронизованный**,**

Шаблон**,** Этот**,** Брось**,** Трэтс/+D2.0+/**,** Истина**,** Пробуй**,** Типдеф**,** Идтипа**,**

Типа**,** Союз**,** Юниттест**,**

Версия**,** Волатайл**,** Пока**,** Для**,**

// Целegral types.

Сим**,** Шим**,** Дим**,** Бул**,**

Байт**,** Ббайт**,** Крат**,** Бкрат**,**

Цел**,** Бцел**,** Дол**,** Бдол**,**

Цент**,** Бцент**,**

Плав**,** Дво**,** Реал**,**

Вплав**,** Вдво**,** Вреал**,**

Кплав**,** Кдво**,** Креал**,** Проц**,**

ГОЛОВА**,** // старт of linked список

КФ**,**

МАКС

**}**

**alias** TOK**.**Абстрактный НачалоКС**;**

**alias** TOK**.**Проц КонецКС**;**

**alias** TOK**.**Сим НачалоИнтегральногоТипа**;**

**alias** TOK**.**Проц КонецИнтегральногоТипа**;**

**alias** TOK**.**ФАЙЛ НачалоСпецСем**;**

**alias** TOK**.**ВЕРСИЯ КонецСпецСем**;**

/// Таблица, преобразующая семы каждого вида в текст.

**const** ткст**[**TOK**.**МАКС**]** семаВТкст **=** **[**

"Неверный"**,**

"Нелегал"**,**

"Комментарий"**,**

"#! /shebang/"**,**

"#line"**,**

`"filespec"`**,**

"НовСтр"**,**

"Пусто"**,**

"Идентификатор"**,**

"Ткст"**,**

"СимЛитерал"**,**

"\_\_FILE\_\_"**,**

"\_\_LINE\_\_"**,**

"\_\_DATE\_\_"**,**

"\_\_TIME\_\_"**,**

"\_\_TIMESTAMP\_\_"**,**

"\_\_VENDOR\_\_"**,**

"\_\_VERSION\_\_"**,**

"Цел32"**,** "Цел64"**,** "Бцел32"**,** "Бцел64"**,**

"Плав32"**,** "Плав64"**,** "Плав80"**,**

"Мнимое32"**,** "Мнимое64"**,** "Мнимое80"**,**

"("**,**

")"**,**

"["**,**

"]"**,**

"{"**,**

"}"**,**

"."**,** ".."**,** "..."**,**

"!<>="**,** // Unordered

"!<>"**,** // UorE

"!<="**,** // UorG

"!<"**,** // UorGorE

"!>="**,** // UorL

"!>"**,** // UorLorE

"<>="**,** // LorEorG

"<>"**,** // LorG

"="**,** "=="**,** "!="**,** "!"**,**

"<="**,** "<"**,**

">="**,** ">"**,**

"<<="**,** "<<"**,**

">>="**,**">>"**,**

">>>="**,** ">>>"**,**

"|="**,** "||"**,** "|"**,**

"&="**,** "&&"**,** "&"**,**

"+="**,** "++"**,** "+"**,**

"-="**,** "--"**,** "-"**,**

"/="**,** "/"**,**

"\*="**,** "\*"**,**

"%="**,** "%"**,**

"^="**,** "^"**,**

"~="**,**

"~"**,**

":"**,**

";"**,**

"?"**,**

","**,**

"$"**,**

"abstract"**,**"alias"**,**"align"**,**"asm"**,**"assert"**,**"auto"**,**"body"**,**

"break"**,**"case"**,**"cast"**,**"catch"**,**

"class"**,**"const"**,**"continue"**,**

"debug"**,**"default"**,**"delegate"**,**"delete"**,**"deprecated"**,**"do"**,**

"else"**,**"enum"**,**"export"**,**"extern"**,**"нет"**,**"final"**,**

"finally"**,**"for"**,**"foreach"**,**"foreach\_reverse"**,**"function"**,**"goto"**,**

"if"**,**"import"**,**"in"**,**"inout"**,**

"interface"**,**"invariant"**,**"is"**,**"lazy"**,**"macro"**,**

"mixin"**,**"module"**,**"new"**,**"nothrow"**,**"null"**,**"out"**,**"override"**,**"package"**,**

"pragma"**,**"private"**,**"protected"**,**"public"**,**"pure"**,**"ref"**,**"return"**,**

"scope"**,**"static"**,**"struct"**,**"super"**,**"switch"**,**"synchronized"**,**

"template"**,**"this"**,**"throw"**,**"\_\_traits"**,**"да"**,**"try"**,**"typedef"**,**"typeid"**,**

"typeof"**,**"union"**,**"unittest"**,**

"version"**,**"volatile"**,**"while"**,**"with"**,**

// Целegral types.

"сим"**,** "шим"**,** "дим"**,** "бул"**,**

"байт"**,** "ббайт"**,** "крат"**,** "бкрат"**,**

"цел"**,** "бцел"**,** "дол"**,** "бдол"**,**

"цент"**,** "бцент"**,**

"плав"**,** "дво"**,** "реал"**,**

"вплав"**,** "вдво"**,** "вреал"**,**

"кплав"**,** "кдво"**,** "креал"**,** "проц "**,**

"ГОЛОВА"**,**

"КФ"

**];**

**static** **assert(**семаВТкст**.**length **==** TOK**.**КФ**+**1**);**

**Парсер (parser)**

**module drc.parser.ImportParser;**

**import** drc**.**parser**.**Parser**;**

**import** drc**.**ast**.**Node**,**

drc**.**ast**.**Declarations**,**

drc**.**ast**.**Statements**;**

**import** drc**.**SourceText**;**

**import** drc**.**Enums**;**

**import** common**;**

**private** **alias** TOK T**;**

/// Облегчённый парсер, который находит лишь инструкции импорта

/// в тексте исходника.

**class** ПарсерИмпорта **:** Парсер

**{**

**this(**ИсходныйТекст исхТекст**)**

**{**

**super(**исхТекст**);**

**}**

**override** СложнаяДекларация старт**()**

**{**

**auto** деклы **=** **new** СложнаяДекларация**;**

**super.**иниц**();**

**if** **(**сема**.**вид **==** T**.**Модуль**)**

деклы **~=** разборДекларацииМодуля**();**

**while** **(**сема**.**вид **!=** T**.**КФ**)**

разборДефиницииДекларации**(**Защита**.**Нет**);**

**return** деклы**;**

**}**

проц разборДефиницииБлокаДеклараций**(**Защита защ**)**

**{**

пропусти**(**T**.**ЛФСкобка**);**

**while** **(**сема**.**вид **!=** T**.**ПФСкобка **&&** сема**.**вид **!=** T**.**КФ**)**

разборДефиницииДекларации**(**защ**);**

пропусти**(**T**.**ПФСкобка**);**

**}**

проц разборБлокаДеклараций**(**Защита защ**)**

**{**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**ЛФСкобка**:**

разборДефиницииБлокаДеклараций**(**защ**);**

**break;**

**case** T**.**Двоеточие**:**

далее**();**

**while** **(**сема**.**вид **!=** T**.**ПФСкобка **&&** сема**.**вид **!=** T**.**КФ**)**

разборДефиницииДекларации**(**защ**);**

**break;**

**default:**

разборДефиницииДекларации**(**защ**);**

**}**

**}**

бул пропускДоЗакрывающего**(**T открывающий**,** T закрывающий**)**

**{**

**alias** сема следщ**;**

бцел уровень **=** 1**;**

**while** **(**1**)**

**{**

лексер**.**возьми**(**следщ**);**

**if** **(**следщ**.**вид **==** открывающий**)**

**++**уровень**;**

**else** **if** **(**следщ**.**вид **==** закрывающий **&&** **--**уровень **==** 0**)**

**return** да**;**

**else** **if** **(**следщ**.**вид **==** T**.**КФ**)**

**break;**

**}**

**return** нет**;**

**}**

проц пропускДоСемыПослеЗакрКСкобки**()**

**{**

пропускДоЗакрывающего**(**T**.**ЛСкобка**,** T**.**ПСкобка**);**

далее**();**

**}**

проц пропускДоСемыПослеЗакрФСкобки**()**

**{**

пропускДоЗакрывающего**(**T**.**ЛФСкобка**,** T**.**ПФСкобка**);**

далее**();**

**}**

проц пропусти**(**TOK лекс**)**

**{**

сема**.**вид **==** лекс **&&** далее**();**

**}**

проц разборАтрибутаЗащиты**()**

**{**

Защита защ**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Приватный**:**

защ **=** Защита**.**Приватный**;** **break;**

**case** T**.**Пакет**:**

защ **=** Защита**.**Пакет**;** **break;**

**case** T**.**Защищённый**:**

защ **=** Защита**.**Защищённый**;** **break;**

**case** T**.**Публичный**:**

защ **=** Защита**.**Публичный**;** **break;**

**case** T**.**Экспорт**:**

защ **=** Защита**.**Экспорт**;** **break;**

**default:**

**assert(**0**);**

**}**

далее**();**

разборБлокаДеклараций**(**защ**);**

**}**

проц разборДефиницииДекларации**(**Защита защ**)**

**{**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Расклад**:**

далее**();**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

далее**(),** далее**(),** далее**();** // ( Integer )

разборБлокаДеклараций**(**защ**);**

**break;**

**case** T**.**Прагма**:**

далее**();**

пропускДоСемыПослеЗакрКСкобки**();**

разборБлокаДеклараций**(**защ**);**

**break;**

**case** T**.**Экспорт**,**

T**.**Приватный**,**

T**.**Пакет**,**

T**.**Защищённый**,**

T**.**Публичный**:**

разборАтрибутаЗащиты**();**

**break;**

// Storage classes

**case** T**.**Экстерн**:**

далее**();**

сема**.**вид **==** T**.**ЛСкобка **&&** пропускДоСемыПослеЗакрКСкобки**();**

разборБлокаДеклараций**(**защ**);**

**break;**

**case** T**.**Конст**:**

**version(**D2**)**

**{**

**if** **(**возьмиСледщ**()** **==** T**.**ЛСкобка**)**

**goto** случай\_Декларация**;**

**}**

**case** T**.**Перепись**,**

T**.**Устаревший**,**

T**.**Абстрактный**,**

T**.**Синхронизованный**,**

// T.Статический,

T**.**Окончательный**,**

T**.**Авто**,**

T**.**Масштаб**:**

случай\_СтатичАтрибут**:**

случай\_АтрибутИнвариант**:**

далее**();**

разборБлокаДеклараций**(**защ**);**

**break;**

// End of storage classes.

**case** T**.**Алиас**,** T**.**Типдеф**:**

далее**();**

**goto** случай\_Декларация**;**

**case** T**.**Статический**:**

**switch** **(**возьмиСледщ**())**

**{**

**case** T**.**Импорт**:**

**goto** случай\_Импорт**;**

**case** T**.**Этот**:**

далее**(),** далее**();** // static this

пропускДоСемыПослеЗакрКСкобки**();**

разборТелаФункции**();**

**break;**

**case** T**.**Тильда**:**

далее**(),** далее**(),** далее**(),** далее**(),** далее**();** // static ~ this ( )

разборТелаФункции**();**

**break;**

**case** T**.**Если**:**

далее**(),** далее**();**

пропускДоСемыПослеЗакрКСкобки**();**

разборБлокаДеклараций**(**защ**);**

**if** **(**сема**.**вид **==** T**.**Иначе**)**

далее**(),** разборБлокаДеклараций**(**защ**);**

**break;**

**case** T**.**Подтвердить**:**

далее**(),** далее**();** // static assert

пропускДоСемыПослеЗакрКСкобки**();**

пропусти**(**T**.**ТочкаЗапятая**);**

**break;**

**default:**

**goto** случай\_СтатичАтрибут**;**

**}**

**break;**

**case** T**.**Импорт**:**

случай\_Импорт**:**

**auto** декл **=** разборДекларацииИмпорта**();**

декл**.**установиЗащиту**(**защ**);** // Set the защита attribute.

импорты **~=** декл**.**в**!(**ДекларацияИмпорта**);**

**break;**

**case** T**.**Перечень**:**

далее**();**

сема**.**вид **==** T**.**Идентификатор **&&** далее**();**

**if** **(**сема**.**вид **==** T**.**Двоеточие**)**

**{**

далее**();**

**while** **(**сема**.**вид **!=** T**.**ЛФСкобка **&&** сема**.**вид **!=** T**.**КФ**)**

далее**();**

**}**

**if** **(**сема**.**вид **==** T**.**ТочкаЗапятая**)**

далее**();**

**else**

пропускДоСемыПослеЗакрФСкобки**();**

**break;**

**case** T**.**Класс**:**

**case** T**.**Интерфейс**:**

далее**(),** пропусти**(**T**.**Идентификатор**);** // class Идентификатор

сема**.**вид **==** T**.**ЛСкобка **&&** пропускДоСемыПослеЗакрКСкобки**();** // Skip template парамы.

**if** **(**сема**.**вид **==** T**.**Двоеточие**)**

**{** // BaseClasses

далее**();**

**while** **(**сема**.**вид **!=** T**.**ЛФСкобка **&&** сема**.**вид **!=** T**.**КФ**)**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)** // Skip ( семы... )

пропускДоСемыПослеЗакрКСкобки**();**

**else**

далее**();**

**}**

**if** **(**сема**.**вид **==** T**.**ТочкаЗапятая**)**

далее**();**

**else**

разборДефиницииБлокаДеклараций**(**Защита**.**Нет**);**

**break;**

**case** T**.**Структура**,** T**.**Союз**:**

далее**();** пропусти**(**T**.**Идентификатор**);**

сема**.**вид **==** T**.**ЛСкобка **&&** пропускДоСемыПослеЗакрКСкобки**();**

**if** **(**сема**.**вид **==** T**.**ТочкаЗапятая**)**

далее**();**

**else**

разборДефиницииБлокаДеклараций**(**Защита**.**Нет**);**

**break;**

**case** T**.**Тильда**:**

далее**();** // ~

**case** T**.**Этот**:**

далее**();** далее**();** далее**();** // this ( )

разборТелаФункции**();**

**break;**

**case** T**.**Инвариант**:**

**version(**D2**)**

**{**

**auto** следщ **=** сема**;**

**if** **(**возьмиПосле**(**следщ**)** **==** T**.**ЛСкобка**)**

**{**

**if** **(**возьмиПосле**(**следщ**)** **!=** T**.**ПСкобка**)**

**goto** случай\_Декларация**;**

**}**

**else**

**goto** случай\_АтрибутИнвариант**;**

**}**

далее**();**

сема**.**вид **==** T**.**ЛСкобка **&&** пропускДоСемыПослеЗакрКСкобки**();**

разборТелаФункции**();**

**break;**

**case** T**.**Юниттест**:**

далее**();**

разборТелаФункции**();**

**break;**

**case** T**.**Отладка**:**

далее**();**

**if** **(**сема**.**вид **==** T**.**Присвоить**)**

**{**

далее**(),** далее**(),** далее**();** // = Condition ;

**break;**

**}**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

далее**(),** далее**(),** далее**();** // ( Condition )

разборБлокаДеклараций**(**защ**);**

**if** **(**сема**.**вид **==** T**.**Иначе**)**

далее**(),** разборБлокаДеклараций**(**защ**);**

**break;**

**case** T**.**Версия**:**

далее**();**

**if** **(**сема**.**вид **==** T**.**Присвоить**)**

**{**

далее**(),** далее**(),** далее**();** // = Condition ;

**break;**

**}**

далее**(),** далее**(),** далее**();** // ( Condition )

разборБлокаДеклараций**(**защ**);**

**if** **(**сема**.**вид **==** T**.**Иначе**)**

далее**(),** разборБлокаДеклараций**(**защ**);**

**break;**

**case** T**.**Шаблон**:**

далее**();**

пропусти**(**T**.**Идентификатор**);**

пропускДоСемыПослеЗакрКСкобки**();**

разборДефиницииБлокаДеклараций**(**Защита**.**Нет**);**

**break;**

**case** T**.**Нов**:**

далее**();**

пропускДоСемыПослеЗакрКСкобки**();**

разборТелаФункции**();**

**break;**

**case** T**.**Удалить**:**

далее**();**

пропускДоСемыПослеЗакрКСкобки**();**

разборТелаФункции**();**

**break;**

**case** T**.**Смесь**:**

**while** **(**сема**.**вид **!=** T**.**ТочкаЗапятая **&&** сема**.**вид **!=** T**.**КФ**)**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

пропускДоСемыПослеЗакрКСкобки**();**

**else**

далее**();**

пропусти**(**T**.**ТочкаЗапятая**);**

**break;**

**case** T**.**ТочкаЗапятая**:**

далее**();**

**break;**

// Декларация

**case** T**.**Идентификатор**,** T**.**Точка**,** T**.**Типа**:**

случай\_Декларация**:**

**while** **(**сема**.**вид **!=** T**.**ТочкаЗапятая **&&** сема**.**вид **!=** T**.**КФ**)**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

пропускДоСемыПослеЗакрКСкобки**();**

**else** **if** **(**сема**.**вид **==** T**.**ЛФСкобка**)**

пропускДоСемыПослеЗакрФСкобки**();**

**else**

далее**();**

пропусти**(**T**.**ТочкаЗапятая**);**

**break;**

**default:**

**if** **(**сема**.**интегральныйТип\_ли**)**

**goto** случай\_Декларация**;**

далее**();**

**}**

**}**

ИнструкцияТелаФункции разборТелаФункции**()**

**{**

**while** **(**1**)**

**{**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**ЛФСкобка**:**

пропускДоСемыПослеЗакрФСкобки**();**

**break;**

**case** T**.**ТочкаЗапятая**:**

далее**();**

**break;**

**case** T**.**Вхо**:**

далее**();**

пропускДоСемыПослеЗакрФСкобки**();**

**continue;**

**case** T**.**Вых**:**

далее**();**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

далее**(),** далее**(),** далее**();** // ( Идентификатор )

пропускДоСемыПослеЗакрФСкобки**();**

**continue;**

**case** T**.**Тело**:**

далее**();**

**goto** **case** T**.**ЛФСкобка**;**

**default:**

**}**

**break;** // Exit loop.

**}**

**return** **null;**

**}**

**}**

/// Author: Aziz Köksal

/// License: GPL3

/// $(Maturity very high)

**module** drc**.**parser**.**Parser**;**

**import** drc**.**lexer**.**Lexer**,**

drc**.**lexer**.**IdTable**;**

**import** drc**.**ast**.**Node**,**

drc**.**ast**.**Declarations**,**

drc**.**ast**.**Statements**,**

drc**.**ast**.**Expressions**,**

drc**.**ast**.**Types**,**

drc**.**ast**.**Parameters**;**

**import** drc**.**Messages**;**

**import** drc**.**Diagnostics**;**

**import** drc**.**Enums**;**

**import** drc**.**CompilerInfo**;**

**import** drc**.**SourceText**;**

**import** drc**.**Unicode**;**

**import** common**;**

**import** core**.**Vararg**;**

/// Парсер производит полный разбор дерева путём исследования

/// списка сем, предоставляемого Лексером.

**class** **Парсер**

**{**

Лексер лексер**;** /// Используется для "лексирования" исходного кода.

Сема**\*** сема**;** /// Текущая непробельная сема.

Сема**\*** предыдущСема**;** /// Предыдущая непробельная сема.

Диагностика диаг**;**

ОшибкаПарсера**[]** ошибки**;** /// Массив сообщений об ошибках парсера.

ДекларацияИмпорта**[]** импорты**;** /// ДекларацииИмпорта в исходном тексте.

/// Атрибуты оцениваются на фазе парсирования.

/// TODO: будет удалено. СемантическаяПроходка1 прозводит обработку атрибутов.

ТипКомпоновки типКомпоновки**;**

Защита защита**;** /// определено

КлассХранения классХранения**;** /// определено

бцел размерРаскладки **=** РАЗМЕР\_РАСКЛАДКИ\_ПО\_УМОЛЧАНИЮ**;** /// определено

**private** **alias** TOK T**;** /// Часто используется данным классом.

**private** **alias** УзелТипа Тип**;**

/// Строит объект Парсер.

/// Параметры:

/// исхТекст = the UTF-8 source код.

/// диаг = используется для сбора сообщений об ошибке.

**this(**ИсходныйТекст исхТекст**,** Диагностика диаг **=** **null)**

**{**

**this.**диаг **=** диаг**;**

лексер **=** **new** Лексер**(**исхТекст**,** диаг**);**

**}**

/// Переходит к первой семе.

**protected** проц иниц**()**

**{**

далее**();**

предыдущСема **=** сема**;**

**}**

/// Переходит к следующей семе.

проц далее**()**

**{**

предыдущСема **=** сема**;**

**do**

**{**

лексер**.**следщСема**();**

сема **=** лексер**.**сема**;**

**}** **while** **(**сема**.**пробел\_ли**)** // Skip whitespace

**}**

/// Запускает парсер и возвращает парсированные декларации.

СложнаяДекларация старт**()**

**{**

иниц**();**

**auto** начало **=** сема**;**

**auto** деклы **=** **new** СложнаяДекларация**;**

**if** **(**сема**.**вид **==** T**.**Модуль**)**

деклы **~=** разборДекларацииМодуля**();**

деклы**.**добавьОпцОтпрыски**(**разборДефиницииДеклараций**());**

установи**(**деклы**,** начало**);**

**return** деклы**;**

**}**

/// Запускает парсер и возвращает парсированные выражения.

Выражение старт2**()**

**{**

иниц**();**

**return** разборВыражения**();**

**}**

// Members related в the method пробуй\_().

бцел пробуем**;** /// Больше than 0 if Парсер is in пробуй\_().

бцел счётОшибок**;** /// Используется для отслеживания числа ошибок при обороте пробуй\_().

/// Этот method executes the delegate методРазбора and when an ошибка occurred

/// the state of the лексер and парсер is restored.

/// Возвращает: the return значение of методРазбора().

ТипИтога пробуй\_**(**ТипИтога**)(**ТипИтога **delegate()** методРазбора**,** **out** бул успех**)**

**{**

// Save члены.

**auto** старСема **=** **this.**сема**;**

**auto** старПредшСема **=** **this.**предыдущСема**;**

**auto** старСчёт **=** **this.**счётОшибок**;**

**++**пробуем**;**

**auto** результат **=** методРазбора**();**

**--**пробуем**;**

// Check if an ошибка occurred.

**if** **(**счётОшибок **!=** старСчёт**)**

**{** // Restore члены.

сема **=** старСема**;**

предыдущСема **=** старПредшСема**;**

лексер**.**сема **=** старСема**;**

счётОшибок **=** старСчёт**;**

успех **=** нет**;**

**}**

**else**

успех **=** да**;**

**return** результат**;**

**}**

/// Вызывает неудачное завершение текущего вызова пробуй\_().

проц провал\_пробы**()**

**{**

**assert(**пробуем**);**

счётОшибок**++;**

**}**

/// Устанавливает начало и конец семы узла синтактического древа.

Класс установи**(**Класс**)(**Класс узел**,** Сема**\*** начало**)**

**{**

узел**.**установиСемы**(**начало**,** **this.**предыдущСема**);**

**return** узел**;**

**}**

///Устанавливает начало и конец семы узла синтактического древа.

Класс установи**(**Класс**)(**Класс узел**,** Сема**\*** начало**,** Сема**\*** конец**)**

**{**

узел**.**установиСемы**(**начало**,** конец**);**

**return** узел**;**

**}**

/// Returns да if установи() has been called on a узел.

**static** бул узелУстановлен**(**Узел узел**)**

**{**

**return** узел**.**начало **!is** **null** **&&** узел**.**конец **!is** **null;**

**}**

/// Возвращает вид следующей семы.

TOK возьмиСледщ**()**

**{**

Сема**\*** следщ **=** сема**;**

**do**

лексер**.**возьми**(**следщ**);**

**while** **(**следщ**.**пробел\_ли**)** // Skip whitespace

**return** следщ**.**вид**;**

**}**

/// Возвращает род семы, следующей за t.

TOK возьмиПосле**(**ref Сема**\*** t**)**

**{**

**assert(**t **!is** **null);**

**do**

лексер**.**возьми**(**t**);**

**while** **(**t**.**пробел\_ли**)** // Skip whitespace

**return** t**.**вид**;**

**}**

/// Проверяет теущую сему на соответствие k по виду и возвращает да.

бул проверено**()(**TOK k**)** // Templatized, so it's inlined.

**{**

**return** сема**.**вид **==** k **?** **(**далее**(),** да**)** **:** нет**;**

**}**

/// Проверяет, чтобы текущая сема была ожидаемого вида,

/// затем приступает за следующую сему.

проц пропусти**()(**TOK ожидаемыйВид**)**

**{**

**assert(**сема**.**вид **==** ожидаемыйВид /+|| \*(цел\*).иниц+/**,** сема**.**исхТекст**());**

далее**();**

**}**

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

| Методы парсинга деклараций |

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

Декларация разборДекларацииМодуля**()**

**{**

**auto** начало **=** сема**;**

пропусти**(**T**.**Модуль**);**

ПКИМодуля пкиМодуля**;**

**do**

пкиМодуля **~=** требуетсяИдентификатор**(**сооб**.**ОжидалсяИдентификаторМодуля**);**

**while** **(**проверено**(**T**.**Точка**))**

требуется**(**T**.**ТочкаЗапятая**);**

**return** установи**(new** ДекларацияМодуля**(**пкиМодуля**),** начало**);**

**}**

/// Парсирует "Дефиниции Деклараций" (определения объявлений) до конца файла.

/// $(PRE

/// DeclDefs :=

/// DeclDef

/// DeclDefs

/// )

Декларация**[]** разборДефиницииДеклараций**()**

**{**

Декларация**[]** деклы**;**

**while** **(**сема**.**вид **!=** T**.**КФ**)**

деклы **~=** разборДефиницииДекларации**();**

**return** деклы**;**

**}**

/// Парсирует тело шаблона, класса, интерфейса структуры или союза.

/// $(PRE

/// DeclDefsBlock :=

/// { }

/// { DeclDefs }

/// )

СложнаяДекларация разборТелаДефиницииДекларации**()**

**{**

// Save attributes.

**auto** типКомпоновки **=** **this.**типКомпоновки**;**

**auto** защита **=** **this.**защита**;**

**auto** классХранения **=** **this.**классХранения**;**

// Clear attributes.

**this.**типКомпоновки **=** ТипКомпоновки**.**Нет**;**

**this.**защита **=** Защита**.**Нет**;**

**this.**классХранения **=** КлассХранения**.**Нет**;**

// Parse body.

**auto** начало **=** сема**;**

**auto** деклы **=** **new** СложнаяДекларация**;**

требуется**(**T**.**ЛФСкобка**);**

**while** **(**сема**.**вид **!=** T**.**ПФСкобка **&&** сема**.**вид **!=** T**.**КФ**)**

деклы **~=** разборДефиницииДекларации**();**

требуетсяЗакрыв**(**T**.**ПФСкобка**,** начало**);**

установи**(**деклы**,** начало**);**

// Restore original значения.

**this.**типКомпоновки **=** типКомпоновки**;**

**this.**защита **=** защита**;**

**this.**классХранения **=** классХранения**;**

**return** деклы**;**

**}**

/// Парсирует ДефиницияДекларации.

Декларация разборДефиницииДекларации**()**

**out(**декл**)**

**{** **assert(**узелУстановлен**(**декл**));** **}**

**body**

**{**

**auto** начало **=** сема**;**

Декларация декл**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Расклад**,**

T**.**Прагма**,**

// Защита attributes

T**.**Экспорт**,**

T**.**Приватный**,**

T**.**Пакет**,**

T**.**Защищённый**,**

T**.**Публичный**:**

декл **=** разборИдентификатораАтрибута**();**

**break;**

// Storage classes

**case** T**.**Экстерн**,**

T**.**Устаревший**,**

T**.**Перепись**,**

T**.**Абстрактный**,**

T**.**Синхронизованный**,**

//T.Статический,

T**.**Окончательный**,**

T**.**Конст**,**

//T.Инвариант, // D 2.0

T**.**Авто**,**

T**.**Масштаб**:**

случай\_СтатичАтрибут**:**

случай\_АтрибутИнвариант**:** // D 2.0

случай\_АтрибутПеречень**:** // D 2.0

**return** разборАтрибутаСохранения**();**

**case** T**.**Алиас**:**

далее**();**

декл **=** **new** ДекларацияАлиаса**(**разборПеременнойИлиФункции**());**

**break;**

**case** T**.**Типдеф**:**

далее**();**

декл **=** **new** ДекларацияТипдефа**(**разборПеременнойИлиФункции**());**

**break;**

**case** T**.**Статический**:**

**switch** **(**возьмиСледщ**())**

**{**

**case** T**.**Импорт**:**

**goto** случай\_Импорт**;**

**case** T**.**Этот**:**

декл **=** разборДекларацииСтатичКонструктора**();**

**break;**

**case** T**.**Тильда**:**

декл **=** разборДекларацииСтатичДеструктора**();**

**break;**

**case** T**.**Если**:**

декл **=** парсируйДекларациюСтатичЕсли**();**

**break;**

**case** T**.**Подтвердить**:**

декл **=** парсируйДекларациюСтатичАссерта**();**

**break;**

**default:**

**goto** случай\_СтатичАтрибут**;**

**}**

**break;**

**case** T**.**Импорт**:**

случай\_Импорт**:**

декл **=** разборДекларацииИмпорта**();**

импорты **~=** декл**.**в**!(**ДекларацияИмпорта**);**

// Handle specially. КлассХранения mustn't be установи.

декл**.**установиЗащиту**(this.**защита**);**

**return** установи**(**декл**,** начало**);**

**case** T**.**Перечень**:**

**version(**D2**)**

**{**

**if** **(**манифестПеречня\_ли**())**

**goto** случай\_АтрибутПеречень**;**

**}**

декл **=** разборДекларацииПеречня**();**

**break;**

**case** T**.**Класс**:**

декл **=** разборДекларацииКласса**();**

**break;**

**case** T**.**Интерфейс**:**

декл **=** разборДекларацииИнтерфейса**();**

**break;**

**case** T**.**Структура**,** T**.**Союз**:**

декл **=** разборДекларацииСтруктурыИлиСоюза**();**

**break;**

**case** T**.**Этот**:**

декл **=** разборДекларацииКонструктора**();**

**break;**

**case** T**.**Тильда**:**

декл **=** разборДекларацииДеструктора**();**

**break;**

**case** T**.**Инвариант**:**

**version(**D2**)**

**{**

**auto** следщ **=** сема**;**

**if** **(**возьмиПосле**(**следщ**)** **==** T**.**ЛСкобка**)**

**{**

**if** **(**возьмиПосле**(**следщ**)** **!=** T**.**ПСкобка**)**

**goto** случай\_Декларация**;** // invariant ( Тип )

**}**

**else**

**goto** случай\_АтрибутИнвариант**;** // invariant as КлассХранения.

**}**

декл **=** разборДекларацииИнварианта**();** // invariant ( )

**break;**

**case** T**.**Юниттест**:**

декл **=** разборДекларацииЮниттеста**();**

**break;**

**case** T**.**Отладка**:**

декл **=** разборДекларацииОтладки**();**

**break;**

**case** T**.**Версия**:**

декл **=** парсируйДекларациюВерсии**();**

**break;**

**case** T**.**Шаблон**:**

декл **=** парсируйДекларациюШаблона**();**

**break;**

**case** T**.**Нов**:**

декл **=** парсируйДекларациюНов**();**

**break;**

**case** T**.**Удалить**:**

декл **=** парсируйДекларациюУдалить**();**

**break;**

**case** T**.**Смесь**:**

декл **=** парсируйМиксин**!(**ДекларацияСмеси**)();**

**break;**

**case** T**.**ТочкаЗапятая**:**

далее**();**

декл **=** **new** ПустаяДекларация**();**

**break;**

// Декларация

**case** T**.**Идентификатор**,** T**.**Точка**,** T**.**Типа**:**

случай\_Декларация**:**

**return** разборПеременнойИлиФункции**(this.**классХранения**,** **this.**защита**,** **this.**типКомпоновки**);**

**default:**

**if** **(**сема**.**интегральныйТип\_ли**)**

**goto** случай\_Декларация**;**

**else** **if** **(**сема**.**вид **==** T**.**Модуль**)**

**{**

декл **=** разборДекларацииМодуля**();**

ошибка**(**начало**,** сооб**.**ДекларацияМодуляНеПервая**);**

**return** декл**;**

**}**

декл **=** **new** НелегальнаяДекларация**();**

// Skip в следщ valid сема.

**do**

далее**();**

**while** **(!**сема**.**началоДефДекл\_ли **&&**

сема**.**вид **!=** T**.**ПФСкобка **&&**

сема**.**вид **!=** T**.**КФ**)**

**auto** текст **=** Сема**.**textSpan**(**начало**,** **this.**предыдущСема**);**

ошибка**(**начало**,** сооб**.**НелегальнаяДекларация**,** текст**);**

**}**

декл**.**установиЗащиту**(this.**защита**);**

декл**.**установиКлассХранения**(this.**классХранения**);**

**assert(!**узелУстановлен**(**декл**));**

установи**(**декл**,** начало**);**

**return** декл**;**

**}**

/// Parses a DeclarationsBlock.

/// $(PRE

/// DeclarationsBlock :=

/// : DeclDefs

/// { }

/// { DeclDefs }

/// DeclDef

/// )

Декларация разборБлокаДеклараций**(**/+бул noДвоеточие = нет+/**)**

**{**

Декларация d**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**ЛФСкобка**:**

**auto** начало **=** сема**;**

далее**();**

**auto** деклы **=** **new** СложнаяДекларация**;**

**while** **(**сема**.**вид **!=** T**.**ПФСкобка **&&** сема**.**вид **!=** T**.**КФ**)**

деклы **~=** разборДефиницииДекларации**();**

требуетсяЗакрыв**(**T**.**ПФСкобка**,** начало**);**

d **=** установи**(**деклы**,** начало**);**

**break;**

**case** T**.**Двоеточие**:**

// if (noДвоеточие == да)

// goto default;

далее**();**

**auto** начало **=** сема**;**

**auto** деклы **=** **new** СложнаяДекларация**;**

**while** **(**сема**.**вид **!=** T**.**ПФСкобка **&&** сема**.**вид **!=** T**.**КФ**)**

деклы **~=** разборДефиницииДекларации**();**

d **=** установи**(**деклы**,** начало**);**

**break;**

**default:**

d **=** разборДефиницииДекларации**();**

**}**

**assert(**узелУстановлен**(**d**));**

**return** d**;**

**}**

// Декларация разборБлокаДекларацийNoДвоеточие()

// {

// return разборБлокаДеклараций(да);

// }

/// Parses either a ДекларацияПеременной or a ДекларацияФункции.

/// Параметры:

/// кхр = предшiously parsed storage classes

/// защита = предшiously parsed защита attribute

/// типКомпоновки = предшiously parsed linkage тип

/// testAutoDeclaration = whether в check for an ДекларацияАвто

/// optionalParameterList = a hint for how в разбор C-style function pointers

Декларация разборПеременнойИлиФункции**(**КлассХранения кхр **=** КлассХранения**.**Нет**,**

Защита защита **=** Защита**.**Нет**,**

ТипКомпоновки типКомпоновки **=** ТипКомпоновки**.**Нет**,**

бул testAutoDeclaration **=** нет**,**

бул optionalParameterList **=** да**)**

**{**

**auto** начало **=** сема**;**

Тип тип**;**

Идентификатор**\*** имя**;**

// Check for ДекларацияАвто: КлассыСохранения Идентификатор =

**if** **(**testAutoDeclaration **&&** сема**.**вид **==** T**.**Идентификатор**)**

**{**

**auto** вид **=** возьмиСледщ**();**

**if** **(**вид **==** T**.**Присвоить**)**

**{** // Авто переменная declaration.

имя **=** сема**.**идент**;**

пропусти**(**T**.**Идентификатор**);**

**goto** LparseVariables**;**

**}**

**else** **version(**D2**)** **if** **(**вид **==** T**.**ЛСкобка**)**

**{** // Check for auto return тип template function.

// КлассыСохранения Name ( TemplateParameterList ) ( ParameterList )

имя **=** сема**.**идент**;**

**auto** следщ **=** сема**;**

возьмиПосле**(**следщ**);**

**if** **(**семаПослеСкобкиЯвляется**(**T**.**ЛСкобка**,** следщ**))**

**{**

пропусти**(**T**.**Идентификатор**);**

**assert(**сема**.**вид **==** T**.**ЛСкобка**);**

**goto** LparseTPList**;** // Далее with parsing a template function.

**}**

**}**

**}**

тип **=** разборТипа**();** // VariableType or ТипИтога

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

**{** // Указатели на функции в стиле Си усложняют грамматику.

// С ними приходится иметь дело отдельно, в масштабе функции.

// Пример:

// проц foo() {

// // Указатель на функцию, принимающий целое число и возвращающий 'some\_type'.

// some\_type (\*p\_func)(цел);

// // Вхо the following case precedence is given в a ВыражениеВызов.

// something(\*p); // 'something' may be a function/method or an объект having opCall overloaded.

// }

// // A pointer в a function taking no параметры and returning 'something'.

// something(\*p);

тип **=** разборТипаУказательНаФункциюСи**(**тип**,** имя**,** optionalParameterList**);**

**}**

**else** **if** **(**возьмиСледщ**()** **==** T**.**ЛСкобка**)**

**{** // Тип FunctionName ( ParameterList ) FunctionBody

имя **=** требуетсяИдентификатор**(**сооб**.**ОжидалосьНазваниеФункции**);**

имя **||** далее**();** // Skip non-identifier сема.

**assert(**сема**.**вид **==** T**.**ЛСкобка**);**

// It's a function declaration

ПараметрыШаблона шпарамы**;**

Выражение констрейнт**;**

**if** **(**семаПослеСкобкиЯвляется**(**T**.**ЛСкобка**))**

LparseTPList**:**

// ( TemplateParameterList ) ( ParameterList )

шпарамы **=** разборСпискаПараметровШаблона**();**

**auto** парамы **=** разборСпискаПараметров**();**

**version(**D2**)**

**{**

**if** **(**шпарамы**)** // Если ( ConstraintExpression )

констрейнт **=** разборДополнительногоКонстрейнта**();**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Конст**:**

кхр **|=** КлассХранения**.**Конст**;**

далее**();**

**break;**

**case** T**.**Инвариант**:**

кхр **|=** КлассХранения**.**Инвариант**;**

далее**();**

**break;**

**default:**

**}**

**}**

// ТипИтога FunctionName ( ParameterList )

**auto** телоФунк **=** разборТелаФункции**();**

**auto** дф **=** **new** ДекларацияФункции**(**тип**,** имя**,**/+ шпарамы,+/ парамы**,** телоФунк**);**

дф**.**установиКлассХранения**(**кхр**);**

дф**.**установиТипКомпоновки**(**типКомпоновки**);**

дф**.**установиЗащиту**(**защита**);**

**if** **(**шпарамы**)**

**{**

**auto** d **=** поместиДекларациюВнутреннегоШаблона**(**начало**,** имя**,** дф**,** шпарамы**,** констрейнт**);**

d**.**установиКлассХранения**(**кхр**);**

d**.**установиЗащиту**(**защита**);**

**return** установи**(**d**,** начало**);**

**}**

**return** установи**(**дф**,** начало**);**

**}**

**else**

**{** // Тип VariableName DeclaratorSuffix

имя **=** требуетсяИдентификатор**(**сооб**.**ОжидалосьНазваниеПеременной**);**

тип **=** разборСуффиксаДекларатора**(**тип**);**

**}**

LparseVariables**:**

// It's a переменные declaration.

Идентификатор**\*[]** имена **=** **[**имя**];** // One identifier has been parsed already.

Выражение**[]** значения**;**

**goto** LenterLoop**;** // Enter the loop and check for an initializer.

**while** **(**проверено**(**T**.**Запятая**))**

**{**

имена **~=** требуетсяИдентификатор**(**сооб**.**ОжидалосьНазваниеПеременной**);**

LenterLoop**:**

**if** **(**проверено**(**T**.**Присвоить**))**

значения **~=** разборИнициализатора**();**

**else**

значения **~=** **null;**

**}**

требуется**(**T**.**ТочкаЗапятая**);**

**auto** d **=** **new** ДекларацияПеременных**(**тип**,** имена**,** значения**);**

d**.**установиКлассХранения**(**кхр**);**

d**.**установиТипКомпоновки**(**типКомпоновки**);**

d**.**установиЗащиту**(**защита**);**

**return** установи**(**d**,** начало**);**

**}**

/// Parses a переменная initializer.

Выражение разборИнициализатора**()**

**{**

**if** **(**сема**.**вид **==** T**.**Проц**)**

**{**

**auto** начало **=** сема**;**

**auto** следщ **=** возьмиСледщ**();**

**if** **(**следщ **==** T**.**Запятая **||** следщ **==** T**.**ТочкаЗапятая**)**

**{**

пропусти**(**T**.**Проц**);**

**return** установи**(new** ВыражениеИницПроц**(),** начало**);**

**}**

**}**

**return** разборНеПроцИнициализатора**();**

**}**

Выражение разборНеПроцИнициализатора**()**

**{**

**auto** начало **=** сема**;**

Выражение иниц**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**ЛКвСкобка**:**

// ArrayInitializer:

// [ ]

// [ ArrayMemberInitializations ]

Выражение**[]** ключи**;**

Выражение**[]** значения**;**

пропусти**(**T**.**ЛКвСкобка**);**

**while** **(**сема**.**вид **!=** T**.**ПКвСкобка**)**

**{**

**auto** в **=** разборНеПроцИнициализатора**();**

**if** **(**проверено**(**T**.**Двоеточие**))**

**{**

ключи **~=** в**;**

значения **~=** разборНеПроцИнициализатора**();**

**}**

**else**

**{**

ключи **~=** **null;**

значения **~=** в**;**

**}**

**if** **(!**проверено**(**T**.**Запятая**))**

**break;**

**}**

требуетсяЗакрыв**(**T**.**ПКвСкобка**,** начало**);**

иниц **=** **new** ВыражениеИницМассива**(**ключи**,** значения**);**

**break;**

**case** T**.**ЛФСкобка**:**

// StructInitializer:

// { }

// { StructMemberInitializers }

Выражение разборИнициализатораСтрукт**()**

**{**

Идентификатор**\*[]** иденты**;**

Выражение**[]** значения**;**

пропусти**(**T**.**ЛФСкобка**);**

**while** **(**сема**.**вид **!=** T**.**ПФСкобка**)**

**{**

**if** **(**сема**.**вид **==** T**.**Идентификатор **&&**

// Peek for colon в see if this is a член identifier.

возьмиСледщ**()** **==** T**.**Двоеточие**)**

**{**

иденты **~=** сема**.**идент**;**

пропусти**(**T**.**Идентификатор**),** пропусти**(**T**.**Двоеточие**);**

**}**

**else**

иденты **~=** **null;**

// NonVoidInitializer

значения **~=** разборНеПроцИнициализатора**();**

**if** **(!**проверено**(**T**.**Запятая**))**

**break;**

**}**

требуетсяЗакрыв**(**T**.**ПФСкобка**,** начало**);**

**return** **new** ВыражениеИницСтруктуры**(**иденты**,** значения**);**

**}**

бул успех**;**

**auto** si **=** пробуй\_**(&**разборИнициализатораСтрукт**,** успех**);**

**if** **(**успех**)**

**{**

иниц **=** si**;**

**break;**

**}**

**assert(**сема**.**вид **==** T**.**ЛФСкобка**);**

//goto default;

**default:**

иниц **=** разборВыраженияПрисвой**();**

**}**

установи**(**иниц**,** начало**);**

**return** иниц**;**

**}**

ИнструкцияТелаФункции разборТелаФункции**()**

**{**

**auto** начало **=** сема**;**

**auto** func **=** **new** ИнструкцияТелаФункции**;**

**while** **(**1**)**

**{**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**ЛФСкобка**:**

func**.**телоФунк **=** разборИнструкций**();**

**break;**

**case** T**.**ТочкаЗапятая**:**

далее**();**

**break;**

**case** T**.**Вхо**:**

**if** **(**func**.**телоВхо**)**

ошибка**(**ИДС**.**КонтрактИн**);**

далее**();**

func**.**телоВхо **=** разборИнструкций**();**

**continue;**

**case** T**.**Вых**:**

**if** **(**func**.**телоВых**)**

ошибка**(**ИДС**.**КонтрактАут**);**

далее**();**

**if** **(**проверено**(**T**.**ЛСкобка**))**

**{**

**auto** leftParen **=** **this.**предыдущСема**;**

func**.**outIdent **=** требуетсяИдентификатор**(**сооб**.**ОжидалсяИдентификатор**);**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

**}**

func**.**телоВых **=** разборИнструкций**();**

**continue;**

**case** T**.**Тело**:**

далее**();**

**goto** **case** T**.**ЛФСкобка**;**

**default:**

ошибка2**(**сооб**.**ОжидалосьТелоФункции**,** сема**);**

**}**

**break;** // Exit loop.

**}**

установи**(**func**,** начало**);**

func**.**завершиКонструкцию**();**

**return** func**;**

**}**

ТипКомпоновки разборТипаКомпоновки**()**

**{**

ТипКомпоновки типКомпоновки**;**

**if** **(!**проверено**(**T**.**ЛСкобка**))**

**return** типКомпоновки**;**

**if** **(**проверено**(**T**.**ПСкобка**))**

**{** // extern()

ошибка**(**ИДС**.**ОтсутствуетТипКомпоновки**);**

**return** типКомпоновки**;**

**}**

**auto** identTok **=** требуетсяИд**();**

ВИД видИд **=** identTok **?** identTok**.**идент**.**видИд **:** ВИД**.**Нуль**;**

**switch** **(**видИд**)**

**{**

**case** ВИД**.**C**:**

**if** **(**проверено**(**T**.**ПлюсПлюс**))**

**{**

типКомпоновки **=** ТипКомпоновки**.**Cpp**;**

**break;**

**}**

типКомпоновки **=** ТипКомпоновки**.**C**;**

**break;**

**case** ВИД**.**D**:**

типКомпоновки **=** ТипКомпоновки**.**D**;**

**break;**

**case** ВИД**.**Windows**:**

типКомпоновки **=** ТипКомпоновки**.**Windows**;**

**break;**

**case** ВИД**.**Pascal**:**

типКомпоновки **=** ТипКомпоновки**.**Pascal**;**

**break;**

**case** ВИД**.**System**:**

типКомпоновки **=** ТипКомпоновки**.**Система**;**

**break;**

**default:**

ошибка2**(**ИДС**.**НеопознанныйТипКомпоновки**,** сема**);**

**}**

требуется**(**T**.**ПСкобка**);**

**return** типКомпоновки**;**

**}**

проц проверьТипКомпоновки**(**ref ТипКомпоновки предш\_тк**,** ТипКомпоновки тк**,** Сема**\*** начало**)**

**{**

**if** **(**предш\_тк **==** ТипКомпоновки**.**Нет**)**

предш\_тк **=** тк**;**

**else**

ошибка**(**начало**,** сооб**.**ПовторяющийсяТипЛинковки**,** Сема**.**textSpan**(**начало**,** **this.**предыдущСема**));**

**}**

Декларация разборАтрибутаСохранения**()**

**{**

КлассХранения кхр**,** stc\_tmp**;**

ТипКомпоновки предш\_типКомпоновки**;**

**auto** saved\_storageClass **=** **this.**классХранения**;** // Save.

// Nested function.

Декларация разбор**()**

**{**

Декларация декл**;**

**auto** начало **=** сема**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Экстерн**:**

**if** **(**возьмиСледщ**()** **!=** T**.**ЛСкобка**)**

**{**

stc\_tmp **=** КлассХранения**.**Экстерн**;**

**goto** Lcommon**;**

**}**

далее**();**

**auto** типКомпоновки **=** разборТипаКомпоновки**();**

проверьТипКомпоновки**(**предш\_типКомпоновки**,** типКомпоновки**,** начало**);**

**auto** saved **=** **this.**типКомпоновки**;** // Save.

**this.**типКомпоновки **=** типКомпоновки**;** // Set.

декл **=** **new** ДекларацияКомпоновки**(**типКомпоновки**,** разбор**());**

установи**(**декл**,** начало**);**

**this.**типКомпоновки **=** saved**;** // Restore.

**break;**

**case** T**.**Перепись**:**

stc\_tmp **=** КлассХранения**.**Перепись**;**

**goto** Lcommon**;**

**case** T**.**Устаревший**:**

stc\_tmp **=** КлассХранения**.**Устаревший**;**

**goto** Lcommon**;**

**case** T**.**Абстрактный**:**

stc\_tmp **=** КлассХранения**.**Абстрактный**;**

**goto** Lcommon**;**

**case** T**.**Синхронизованный**:**

stc\_tmp **=** КлассХранения**.**Синхронизованный**;**

**goto** Lcommon**;**

**case** T**.**Статический**:**

stc\_tmp **=** КлассХранения**.**Статический**;**

**goto** Lcommon**;**

**case** T**.**Окончательный**:**

stc\_tmp **=** КлассХранения**.**Окончательный**;**

**goto** Lcommon**;**

**case** T**.**Конст**:**

**version(**D2**)**

**{**

**if** **(**возьмиСледщ**()** **==** T**.**ЛСкобка**)**

**goto** случай\_Декларация**;**

**}**

stc\_tmp **=** КлассХранения**.**Конст**;**

**goto** Lcommon**;**

**version(**D2**)**

**{**

**case** T**.**Инвариант**:** // D 2.0

**auto** следщ **=** сема**;**

**if** **(**возьмиПосле**(**следщ**)** **==** T**.**ЛСкобка**)**

**{**

**if** **(**возьмиПосле**(**следщ**)** **!=** T**.**ПСкобка**)**

**goto** случай\_Декларация**;** // invariant ( Тип )

декл **=** разборДекларацииИнварианта**();** // invariant ( )

// NB: this must be similar в the код at the конец of

// разборДефиницииДекларации().

декл**.**установиЗащиту**(this.**защита**);**

декл**.**установиКлассХранения**(**кхр**);**

установи**(**декл**,** начало**);**

**break;**

**}**

// инвариант как классХранения.

stc\_tmp **=** КлассХранения**.**Инвариант**;**

**goto** Lcommon**;**

**case** T**.**Перечень**:** // D 2.0

**if** **(!**манифестПеречня\_ли**())**

**{** // A normal enum declaration.

декл **=** разборДекларацииПеречня**();**

// NB: this must be similar в the код at the конец of

// разборДефиницииДекларации().

декл**.**установиЗащиту**(this.**защита**);**

декл**.**установиКлассХранения**(**кхр**);**

установи**(**декл**,** начало**);**

**break;**

**}**

// enum as КлассХранения.

stc\_tmp **=** КлассХранения**.**Манифест**;**

**goto** Lcommon**;**

**}** // version(D2)

**case** T**.**Авто**:**

stc\_tmp **=** КлассХранения**.**Авто**;**

**goto** Lcommon**;**

**case** T**.**Масштаб**:**

stc\_tmp **=** КлассХранения**.**Масштаб**;**

**goto** Lcommon**;**

Lcommon**:**

// Issue ошибка if redundant.

**if** **(**кхр **&** stc\_tmp**)**

ошибка2**(**ИДС**.**ПовторяющийсяКлассХранения**,** сема**);**

**else**

кхр **|=** stc\_tmp**;**

далее**();**

декл **=** **new** ДекларацияКлассаХранения**(**stc\_tmp**,** разбор**());**

установи**(**декл**,** начало**);**

**break;**

**case** T**.**Идентификатор**:**

случай\_Декларация**:**

// Этот could be a normal Декларация or an ДекларацияАвто

декл **=** разборПеременнойИлиФункции**(**кхр**,** **this.**защита**,** предш\_типКомпоновки**,** да**);**

**break;**

**default:**

**this.**классХранения **=** кхр**;** // Set.

декл **=** разборБлокаДеклараций**();**

**this.**классХранения **=** saved\_storageClass**;** // Reset.

**}**

**assert(**узелУстановлен**(**декл**));**

**return** декл**;**

**}**

**return** разбор**();**

**}**

бцел разборАтрибутаАлайн**()**

**{**

пропусти**(**T**.**Расклад**);**

бцел размер **=** РАЗМЕР\_РАСКЛАДКИ\_ПО\_УМОЛЧАНИЮ**;** // Global default.

**if** **(**проверено**(**T**.**ЛСкобка**))**

**{**

**if** **(**сема**.**вид **==** T**.**Цел32**)**

**(**размер **=** сема**.**цел\_**),** пропусти**(**T**.**Цел32**);**

**else**

ожидаемое**(**T**.**Цел32**);**

требуется**(**T**.**ПСкобка**);**

**}**

**return** размер**;**

**}**

Декларация разборИдентификатораАтрибута**()**

**{**

Декларация декл**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Расклад**:**

бцел размерРаскладки **=** разборАтрибутаАлайн**();**

**auto** saved **=** **this.**размерРаскладки**;** // Save.

**this.**размерРаскладки **=** размерРаскладки**;** // Set.

декл **=** **new** ДекларацияРазложи**(**размерРаскладки**,** разборБлокаДеклараций**());**

**this.**размерРаскладки **=** saved**;** // Restore.

**break;**

**case** T**.**Прагма**:**

// Прагма:

// pragma ( Идентификатор )

// pragma ( Идентификатор , ExpressionList )

далее**();**

Идентификатор**\*** идент**;**

Выражение**[]** арги**;**

требуется**(**T**.**ЛСкобка**);**

идент **=** требуетсяИдентификатор**(**сооб**.**ОжидалсяИдентификаторПрагмы**);**

**if** **(**проверено**(**T**.**Запятая**))**

арги **=** разборСпискаВыражений**();**

требуется**(**T**.**ПСкобка**);**

декл **=** **new** ДекларацияПрагмы**(**идент**,** арги**,** разборБлокаДеклараций**());**

**break;**

**default:**

// Защита attributes

Защита защ**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Приватный**:**

защ **=** Защита**.**Приватный**;** **break;**

**case** T**.**Пакет**:**

защ **=** Защита**.**Пакет**;** **break;**

**case** T**.**Защищённый**:**

защ **=** Защита**.**Защищённый**;** **break;**

**case** T**.**Публичный**:**

защ **=** Защита**.**Публичный**;** **break;**

**case** T**.**Экспорт**:**

защ **=** Защита**.**Экспорт**;** **break;**

**default:**

**assert(**0**);**

**}**

далее**();**

**auto** saved **=** **this.**защита**;** // Save.

**this.**защита **=** защ**;** // Set.

декл **=** **new** ДекларацияЗащиты**(**защ**,** разборБлокаДеклараций**());**

**this.**защита **=** saved**;** // Restore.

**}**

**return** декл**;**

**}**

Декларация разборДекларацииИмпорта**()**

**{**

бул статический\_ли **=** проверено**(**T**.**Статический**);**

пропусти**(**T**.**Импорт**);**

ПКИМодуля**[]** пкиМодулей**;**

Идентификатор**\*[]** алиасыМодуля**;**

Идентификатор**\*[]** связанныеИмена**;**

Идентификатор**\*[]** связанныеАлиасы**;**

**do**

**{**

ПКИМодуля пкиМодуля**;**

Идентификатор**\*** moduleAlias**;**

// AliasName = ModuleName

**if** **(**возьмиСледщ**()** **==** T**.**Присвоить**)**

**{**

moduleAlias **=** требуетсяИдентификатор**(**сооб**.**ExpectedAliasModuleName**);**

пропусти**(**T**.**Присвоить**);**

**}**

// Идентификатор ("." Идентификатор)\*

**do**

пкиМодуля **~=** требуетсяИдентификатор**(**сооб**.**ОжидалсяИдентификаторМодуля**);**

**while** **(**проверено**(**T**.**Точка**))**

// Push identifiers.

пкиМодулей **~=** пкиМодуля**;**

алиасыМодуля **~=** moduleAlias**;**

**}** **while** **(**проверено**(**T**.**Запятая**))**

**if** **(**проверено**(**T**.**Двоеточие**))**

**{** // BindAlias "=" BindName ("," BindAlias "=" BindName)\*;

// BindName ("," BindName)\*;

**do**

**{**

Идентификатор**\*** bindAlias**;**

// BindAlias = BindName

**if** **(**возьмиСледщ**()** **==** T**.**Присвоить**)**

**{**

bindAlias **=** требуетсяИдентификатор**(**сооб**.**ОжидаемоеИмяАлиасаИмпорта**);**

пропусти**(**T**.**Присвоить**);**

**}**

// Push identifiers.

связанныеИмена **~=** требуетсяИдентификатор**(**сооб**.**ОжидаемоеИмяИмпорта**);**

связанныеАлиасы **~=** bindAlias**;**

**}** **while** **(**проверено**(**T**.**Запятая**))**

**}**

требуется**(**T**.**ТочкаЗапятая**);**

**return** **new** ДекларацияИмпорта**(**пкиМодулей**,** алиасыМодуля**,** связанныеИмена**,** связанныеАлиасы**,** статический\_ли**);**

**}**

**version(**D2**)**

**{**

/// Возвращает да, если из\_ an enum manifest or

/// нет if it's a normal enum declaration.

бул манифестПеречня\_ли**()**

**{**

**assert(**сема**.**вид **==** T**.**Перечень**);**

**auto** следщ **=** сема**;**

**auto** вид **=** возьмиПосле**(**следщ**);**

**if** **(**вид **==** T**.**Двоеточие **||** вид **==** T**.**ЛФСкобка**)**

**return** нет**;** // Anonymous enum.

**else** **if** **(**вид **==** T**.**Идентификатор**)**

**{**

вид **=** возьмиПосле**(**следщ**);**

**if** **(**вид **==** T**.**Двоеточие **||** вид **==** T**.**ЛФСкобка **||** вид **==** T**.**ТочкаЗапятая**)**

**return** нет**;** // Named enum.

**}**

**return** да**;** // Манифест enum.

**}**

**}**

Декларация разборДекларацииПеречня**()**

**{**

пропусти**(**T**.**Перечень**);**

Идентификатор**\*** имяПеречня**;**

Тип типОснова**;**

ДекларацияЧленаПеречня**[]** члены**;**

бул естьТело**;**

имяПеречня **=** дополнительныйИдентификатор**();**

**if** **(**проверено**(**T**.**Двоеточие**))**

типОснова **=** разборБазовогоТипа**();**

**if** **(**имяПеречня **&&** проверено**(**T**.**ТочкаЗапятая**))**

**{}**

**else** **if** **(**проверено**(**T**.**ЛФСкобка**))**

**{**

**auto** леваяФСкобка **=** **this.**предыдущСема**;**

естьТело **=** да**;**

**while** **(**сема**.**вид **!=** T**.**ПФСкобка**)**

**{**

**auto** начало **=** сема**;**

Тип тип**;**

**version(**D2**)**

**{**

бул успех**;**

пробуй\_**({**

// Тип Идентификатор = ВыражениеПрисвой

тип **=** разборТипа**();** // Set outer тип переменная.

**if** **(**сема**.**вид **!=** T**.**Идентификатор**)**

провал\_пробы**(),** **(**тип **=** **null);**

**return** **null;**

**},** успех**);**

**}**

**auto** имя **=** требуетсяИдентификатор**(**сооб**.**ОжидалсяЧленПеречня**);**

Выражение значение**;**

**if** **(**проверено**(**T**.**Присвоить**))**

значение **=** разборВыраженияПрисвой**();**

члены **~=** установи**(new** ДекларацияЧленаПеречня**(**тип**,** имя**,** значение**),** начало**);**

**if** **(!**проверено**(**T**.**Запятая**))**

**break;**

**}**

требуетсяЗакрыв**(**T**.**ПФСкобка**,** леваяФСкобка**);**

**}**

**else**

ошибка2**(**сооб**.**ОжидалосьТелоПеречня**,** сема**);**

**return** **new** ДекларацияПеречня**(**имяПеречня**,** типОснова**,** члены**,** естьТело**);**

**}**

/// Wraps a declaration внутри a template declaration.

/// Параметры:

/// начало = начало сема of декл.

/// имя = имя of декл.

/// декл = the declaration в be wrapped.

/// шпарамы = the template параметры.

/// констрейнт = the констрейнт выражение.

ДекларацияШаблона поместиДекларациюВнутреннегоШаблона**(**Сема**\*** начало**,**

Идентификатор**\*** имя**,**

Декларация декл**,**

ПараметрыШаблона шпарамы**,**

Выражение констрейнт**)**

**{**

установи**(**декл**,** начало**);**

**auto** cd **=** **new** СложнаяДекларация**;**

cd **~=** декл**;**

установи**(**cd**,** начало**);**

**return** **new** ДекларацияШаблона**(**имя**,** шпарамы**,** констрейнт**,** cd**);**

**}**

Декларация разборДекларацииКласса**()**

**{**

**auto** начало **=** сема**;**

пропусти**(**T**.**Класс**);**

Идентификатор**\*** имяКласса**;**

ПараметрыШаблона шпарамы**;**

Выражение констрейнт**;**

ТипКлассОснова**[]** основы**;**

СложнаяДекларация деклы**;**

имяКласса **=** требуетсяИдентификатор**(**сооб**.**ОжидалосьНазваниеКласса**);**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

**{**

шпарамы **=** разборСпискаПараметровШаблона**();**

**version(**D2**)** констрейнт **=** разборДополнительногоКонстрейнта**();**

**}**

**if** **(**сема**.**вид **==** T**.**Двоеточие**)**

основы **=** разборБазовыхКлассов**();**

**if** **(**основы**.**length **==** 0 **&&** проверено**(**T**.**ТочкаЗапятая**))**

**{}**

**else** **if** **(**сема**.**вид **==** T**.**ЛФСкобка**)**

деклы **=** разборТелаДефиницииДекларации**();**

**else**

ошибка2**(**сооб**.**ОжидалосьТелоКласса**,** сема**);**

Декларация d **=** **new** ДекларацияКласса**(**имяКласса**,** /+шпарамы, +/основы**,** деклы**);**

**if** **(**шпарамы**)**

d **=** поместиДекларациюВнутреннегоШаблона**(**начало**,** имяКласса**,** d**,** шпарамы**,** констрейнт**);**

**return** d**;**

**}**

ТипКлассОснова**[]** разборБазовыхКлассов**(**бул colonLeadsOff **=** да**)**

**{**

colonLeadsOff **&&** пропусти**(**T**.**Двоеточие**);**

ТипКлассОснова**[]** основы**;**

**do**

**{**

Защита защ **=** Защита**.**Публичный**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Идентификатор**,** T**.**Точка**,** T**.**Типа**:** **goto** LparseBasicType**;**

**case** T**.**Приватный**:** защ **=** Защита**.**Приватный**;** **break;**

**case** T**.**Защищённый**:** защ **=** Защита**.**Защищённый**;** **break;**

**case** T**.**Пакет**:** защ **=** Защита**.**Пакет**;** **break;**

**case** T**.**Публичный**:** /\*защ = Защита.Публичный;\*/ **break;**

**default:**

ошибка2**(**ИДС**.**ExpectedBaseClasses**,** сема**);**

**return** основы**;**

**}**

далее**();** // Skip защита attribute.

LparseBasicType**:**

**auto** начало **=** сема**;**

**auto** тип **=** разборБазовогоТипа**();**

основы **~=** установи**(new** ТипКлассОснова**(**защ**,** тип**),** начало**);**

**}** **while** **(**проверено**(**T**.**Запятая**))**

**return** основы**;**

**}**

Декларация разборДекларацииИнтерфейса**()**

**{**

**auto** начало **=** сема**;**

пропусти**(**T**.**Интерфейс**);**

Идентификатор**\*** имя**;**

ПараметрыШаблона шпарамы**;**

Выражение констрейнт**;**

ТипКлассОснова**[]** основы**;**

СложнаяДекларация деклы**;**

имя **=** требуетсяИдентификатор**(**сооб**.**ОжидалосьНазваниеИнтерфейса**);**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

**{**

шпарамы **=** разборСпискаПараметровШаблона**();**

**version(**D2**)** констрейнт **=** разборДополнительногоКонстрейнта**();**

**}**

**if** **(**сема**.**вид **==** T**.**Двоеточие**)**

основы **=** разборБазовыхКлассов**();**

**if** **(**основы**.**length **==** 0 **&&** проверено**(**T**.**ТочкаЗапятая**))**

**{}**

**else** **if** **(**сема**.**вид **==** T**.**ЛФСкобка**)**

деклы **=** разборТелаДефиницииДекларации**();**

**else**

ошибка2**(**сооб**.**ОжидалосьТелоИнтерфейса**,** сема**);**

Декларация d **=** **new** ДекларацияИнтерфейса**(**имя**,** /+шпарамы, +/основы**,** деклы**);**

**if** **(**шпарамы**)**

d **=** поместиДекларациюВнутреннегоШаблона**(**начало**,** имя**,** d**,** шпарамы**,** констрейнт**);**

**return** d**;**

**}**

Декларация разборДекларацииСтруктурыИлиСоюза**()**

**{**

**assert(**сема**.**вид **==** T**.**Структура **||** сема**.**вид **==** T**.**Союз**);**

**auto** начало **=** сема**;**

пропусти**(**сема**.**вид**);**

Идентификатор**\*** имя**;**

ПараметрыШаблона шпарамы**;**

Выражение констрейнт**;**

СложнаяДекларация деклы**;**

имя **=** дополнительныйИдентификатор**();**

**if** **(**имя **&&** сема**.**вид **==** T**.**ЛСкобка**)**

**{**

шпарамы **=** разборСпискаПараметровШаблона**();**

**version(**D2**)** констрейнт **=** разборДополнительногоКонстрейнта**();**

**}**

**if** **(**имя **&&** проверено**(**T**.**ТочкаЗапятая**))**

**{}**

**else** **if** **(**сема**.**вид **==** T**.**ЛФСкобка**)**

деклы **=** разборТелаДефиницииДекларации**();**

**else**

ошибка2**(**начало**.**вид **==** T**.**Структура **?**

сооб**.**ОжидалосьТелоСтруктуры **:**

сооб**.**ОжидалосьТелоСоюза**,** сема**);**

Декларация d**;**

**if** **(**начало**.**вид **==** T**.**Структура**)**

**{**

**auto** sd **=** **new** ДекларацияСтруктуры**(**имя**,** /+шпарамы, +/деклы**);**

sd**.**установиРазмерРаскладки**(this.**размерРаскладки**);**

d **=** sd**;**

**}**

**else**

d **=** **new** ДекларацияСоюза**(**имя**,** /+шпарамы, +/деклы**);**

**if** **(**шпарамы**)**

d **=** поместиДекларациюВнутреннегоШаблона**(**начало**,** имя**,** d**,** шпарамы**,** констрейнт**);**

**return** d**;**

**}**

Декларация разборДекларацииКонструктора**()**

**{**

пропусти**(**T**.**Этот**);**

**auto** параметры **=** разборСпискаПараметров**();**

**auto** телоФунк **=** разборТелаФункции**();**

**return** **new** ДекларацияКонструктора**(**параметры**,** телоФунк**);**

**}**

Декларация разборДекларацииДеструктора**()**

**{**

пропусти**(**T**.**Тильда**);**

требуется**(**T**.**Этот**);**

требуется**(**T**.**ЛСкобка**);**

требуется**(**T**.**ПСкобка**);**

**auto** телоФунк **=** разборТелаФункции**();**

**return** **new** ДекларацияДеструктора**(**телоФунк**);**

**}**

Декларация разборДекларацииСтатичКонструктора**()**

**{**

пропусти**(**T**.**Статический**);**

пропусти**(**T**.**Этот**);**

требуется**(**T**.**ЛСкобка**);**

требуется**(**T**.**ПСкобка**);**

**auto** телоФунк **=** разборТелаФункции**();**

**return** **new** ДекларацияСтатическогоКонструктора**(**телоФунк**);**

**}**

Декларация разборДекларацииСтатичДеструктора**()**

**{**

пропусти**(**T**.**Статический**);**

пропусти**(**T**.**Тильда**);**

требуется**(**T**.**Этот**);**

требуется**(**T**.**ЛСкобка**);**

требуется**(**T**.**ПСкобка**);**

**auto** телоФунк **=** разборТелаФункции**();**

**return** **new** ДекларацияСтатическогоДеструктора**(**телоФунк**);**

**}**

Декларация разборДекларацииИнварианта**()**

**{**

пропусти**(**T**.**Инвариант**);**

// Optional () for getting ready porting в D 2.0

**if** **(**проверено**(**T**.**ЛСкобка**))**

требуется**(**T**.**ПСкобка**);**

**auto** телоФунк **=** разборТелаФункции**();**

**return** **new** ДекларацияИнварианта**(**телоФунк**);**

**}**

Декларация разборДекларацииЮниттеста**()**

**{**

пропусти**(**T**.**Юниттест**);**

**auto** телоФунк **=** разборТелаФункции**();**

**return** **new** ДекларацияЮниттеста**(**телоФунк**);**

**}**

Сема**\*** разборИдентИлиЦел**()**

**{**

**if** **(**проверено**(**T**.**Цел32**)** **||** проверено**(**T**.**Идентификатор**))**

**return** **this.**предыдущСема**;**

ошибка2**(**сооб**.**ОжидалсяИдентИлиЦел**,** сема**);**

**return** **null;**

**}**

Сема**\*** разборУсловияВерсии**()**

**{**

**version(**D2**)**

**{**

**if** **(**проверено**(**T**.**Юниттест**))**

**return** **this.**предыдущСема**;**

**}**

**return** разборИдентИлиЦел**();**

**}**

Декларация разборДекларацииОтладки**()**

**{**

пропусти**(**T**.**Отладка**);**

Сема**\*** спец**;**

Сема**\*** услов**;**

Декларация деклы**,** деклыИначе**;**

**if** **(**проверено**(**T**.**Присвоить**))**

**{** // debug = Integer ;

// debug = Идентификатор ;

спец **=** разборИдентИлиЦел**();**

требуется**(**T**.**ТочкаЗапятая**);**

**}**

**else**

**{** // ( Condition )

**if** **(**проверено**(**T**.**ЛСкобка**))**

**{**

услов **=** разборИдентИлиЦел**();**

требуется**(**T**.**ПСкобка**);**

**}**

// debug DeclarationsBlock

// debug ( Condition ) DeclarationsBlock

деклы **=** разборБлокаДеклараций**();**

// else DeclarationsBlock

**if** **(**проверено**(**T**.**Иначе**))**

деклыИначе **=** разборБлокаДеклараций**();**

**}**

**return** **new** ДекларацияОтладки**(**спец**,** услов**,** деклы**,** деклыИначе**);**

**}**

Декларация парсируйДекларациюВерсии**()**

**{**

пропусти**(**T**.**Версия**);**

Сема**\*** спец**;**

Сема**\*** услов**;**

Декларация деклы**,** деклыИначе**;**

**if** **(**проверено**(**T**.**Присвоить**))**

**{** // version = Integer ;

// version = Идентификатор ;

спец **=** разборИдентИлиЦел**();**

требуется**(**T**.**ТочкаЗапятая**);**

**}**

**else**

**{** // ( Condition )

требуется**(**T**.**ЛСкобка**);**

услов **=** разборУсловияВерсии**();**

требуется**(**T**.**ПСкобка**);**

// version ( Condition ) DeclarationsBlock

деклы **=** разборБлокаДеклараций**();**

// else DeclarationsBlock

**if** **(**проверено**(**T**.**Иначе**))**

деклыИначе **=** разборБлокаДеклараций**();**

**}**

**return** **new** ДекларацияВерсии**(**спец**,** услов**,** деклы**,** деклыИначе**);**

**}**

Декларация парсируйДекларациюСтатичЕсли**()**

**{**

пропусти**(**T**.**Статический**);**

пропусти**(**T**.**Если**);**

Выражение условие**;**

Декларация деклыЕсли**,** деклыИначе**;**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

условие **=** разборВыраженияПрисвой**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

деклыЕсли **=** разборБлокаДеклараций**();**

**if** **(**проверено**(**T**.**Иначе**))**

деклыИначе **=** разборБлокаДеклараций**();**

**return** **new** ДекларацияСтатическогоЕсли**(**условие**,** деклыЕсли**,** деклыИначе**);**

**}**

Декларация парсируйДекларациюСтатичАссерта**()**

**{**

пропусти**(**T**.**Статический**);**

пропусти**(**T**.**Подтвердить**);**

Выражение условие**,** сообщение**;**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

условие **=** разборВыраженияПрисвой**();**

**if** **(**проверено**(**T**.**Запятая**))**

сообщение **=** разборВыраженияПрисвой**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

требуется**(**T**.**ТочкаЗапятая**);**

**return** **new** ДекларацияСтатическогоПодтверди**(**условие**,** сообщение**);**

**}**

Декларация парсируйДекларациюШаблона**()**

**{**

пропусти**(**T**.**Шаблон**);**

**auto** имя **=** требуетсяИдентификатор**(**сооб**.**ОжидалосьНазваниеШаблона**);**

**auto** шпарамы **=** разборСпискаПараметровШаблона**();**

**auto** констрейнт **=** разборДополнительногоКонстрейнта**();**

**auto** деклы **=** разборТелаДефиницииДекларации**();**

**return** **new** ДекларацияШаблона**(**имя**,** шпарамы**,** констрейнт**,** деклы**);**

**}**

Декларация парсируйДекларациюНов**()**

**{**

пропусти**(**T**.**Нов**);**

**auto** параметры **=** разборСпискаПараметров**();**

**auto** телоФунк **=** разборТелаФункции**();**

**return** **new** ДекларацияНов**(**параметры**,** телоФунк**);**

**}**

Декларация парсируйДекларациюУдалить**()**

**{**

пропусти**(**T**.**Удалить**);**

**auto** параметры **=** разборСпискаПараметров**();**

**auto** телоФунк **=** разборТелаФункции**();**

**return** **new** ДекларацияУдали**(**параметры**,** телоФунк**);**

**}**

Тип parseTypeofType**()**

**{**

**auto** начало **=** сема**;**

пропусти**(**T**.**Типа**);**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

Тип тип**;**

**switch** **(**сема**.**вид**)**

**{**

**version(**D2**)**

**{**

**case** T**.**Итог**:**

далее**();**

тип **=** **new** ТТип**();**

**break;**

**}**

**default:**

тип **=** **new** ТТип**(**разборВыражения**());**

**}**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

установи**(**тип**,** начало**);**

**return** тип**;**

**}**

/// Parses a ДекларацияСмеси or ИнструкцияСмесь.

/// $(PRE

/// TemplateMixin :=

/// mixin ( ВыражениеПрисвой ) ;

/// mixin TemplateIdentifier ;

/// mixin TemplateIdentifier MixinIdentifier ;

/// mixin TemplateIdentifier !( АргументыШаблона ) ;

/// mixin TemplateIdentifier !( АргументыШаблона ) MixinIdentifier ;

/// )

Класс парсируйМиксин**(**Класс**)()**

**{**

**static** **assert(is(**Класс **==** ДекларацияСмеси**)** **||** **is(**Класс **==** ИнструкцияСмесь**));**

пропусти**(**T**.**Смесь**);**

**static** **if** **(is(**Класс **==** ДекларацияСмеси**))**

**{**

**if** **(**проверено**(**T**.**ЛСкобка**))**

**{**

**auto** leftParen **=** сема**;**

**auto** в **=** разборВыраженияПрисвой**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

требуется**(**T**.**ТочкаЗапятая**);**

**return** **new** ДекларацияСмеси**(**в**);**

**}**

**}**

**auto** начало **=** сема**;**

Выражение в**;**

Идентификатор**\*** идентСмеси**;**

**if** **(**проверено**(**T**.**Точка**))**

в **=** установи**(new** ВыражениеМасштабМодуля**(**разборВыраженияИдентификатора**()),** начало**);**

**else**

в **=** разборВыраженияИдентификатора**();**

**while** **(**проверено**(**T**.**Точка**))**

в **=** установи**(new** ВыражениеТочка**(**в**,** разборВыраженияИдентификатора**()),** начало**);**

идентСмеси **=** дополнительныйИдентификатор**();**

требуется**(**T**.**ТочкаЗапятая**);**

**return** **new** Класс**(**в**,** идентСмеси**);**

**}**

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

| Инструкция parsing methods |

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

СложнаяИнструкция разборИнструкций**()**

**{**

**auto** начало **=** сема**;**

требуется**(**T**.**ЛФСкобка**);**

**auto** инструкции **=** **new** СложнаяИнструкция**();**

**while** **(**сема**.**вид **!=** T**.**ПФСкобка **&&** сема**.**вид **!=** T**.**КФ**)**

инструкции **~=** разборИнструкции**();**

требуетсяЗакрыв**(**T**.**ПФСкобка**,** начало**);**

**return** установи**(**инструкции**,** начало**);**

**}**

/// Parses a Инструкция.

Инструкция разборИнструкции**()**

**{**

**auto** начало **=** сема**;**

Инструкция s**;**

Декларация d**;**

**if** **(**сема**.**интегральныйТип\_ли**)**

**{**

d **=** разборПеременнойИлиФункции**();**

**goto** LreturnDeclarationStatement**;**

**}**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Расклад**:**

бцел размер **=** разборАтрибутаАлайн**();**

// Reтктict align attribute в structs in parsing phase.

ДекларацияСтруктуры structDecl**;**

**if** **(**сема**.**вид **==** T**.**Структура**)**

**{**

**auto** begin2 **=** сема**;**

structDecl **=** разборДекларацииСтруктурыИлиСоюза**().**в**!(**ДекларацияСтруктуры**);**

structDecl**.**установиРазмерРаскладки**(**размер**);**

установи**(**structDecl**,** begin2**);**

**}**

**else**

ожидаемое**(**T**.**Структура**);**

d **=** **new** ДекларацияРазложи**(**размер**,** structDecl **?** **cast(**Декларация**)**structDecl **:** **new** СложнаяДекларация**);**

**goto** LreturnDeclarationStatement**;**

/+ Не applicable for инструкции.

T.Приватный, T.Пакет, T.Защищённый, T.Публичный, T.Экспорт,

T.Устаревший, T.Перепись, T.Абстрактный,+/

**case** T**.**Экстерн**,**

T**.**Окончательный**,**

T**.**Конст**,**

T**.**Авто**:**

//T.Масштаб

//T.Статический

случай\_разборАтрибута**:**

s **=** разборИнструкцииАтрибута**();**

**return** s**;**

**case** T**.**Идентификатор**:**

**if** **(**возьмиСледщ**()** **==** T**.**Двоеточие**)**

**{**

**auto** идент **=** сема**.**идент**;**

пропусти**(**T**.**Идентификатор**);** пропусти**(**T**.**Двоеточие**);**

s **=** **new** ИнструкцияСМеткой**(**идент**,** разборИнструкцииБезМасштабаИлиПустое**());**

**break;**

**}**

**goto** **case** T**.**Точка**;**

**case** T**.**Точка**,** T**.**Типа**:**

бул успех**;**

d **=** пробуй\_**(delegate** **{**

**return** разборПеременнойИлиФункции**(**КлассХранения**.**Нет**,**

Защита**.**Нет**,**

ТипКомпоновки**.**Нет**,** нет**,** нет**);**

**},** успех

**);**

**if** **(**успех**)**

**goto** LreturnDeclarationStatement**;** // Декларация

**else**

**goto** случай\_разборИнструкцииВыражения**;** // Выражение

**case** T**.**Если**:**

s **=** разборИнструкцииЕсли**();**

**break;**

**case** T**.**Пока**:**

s **=** разборИнструкцииПока**();**

**break;**

**case** T**.**Делай**:**

s **=** разборИнструкцииДелайПока**();**

**break;**

**case** T**.**При**:**

s **=** разборИнструкцииПри**();**

**break;**

**case** T**.**Длявсех**,** T**.**Длявсех\_реверс**:**

s **=** разборИнструкцииДлявсех**();**

**break;**

**case** T**.**Щит**:**

s **=** разборИнструкцииЩит**();**

**break;**

**case** T**.**Реле**:**

s **=** разборИнструкцииРеле**();**

**break;**

**case** T**.**Дефолт**:**

s **=** разборИнструкцииДефолт**();**

**break;**

**case** T**.**Далее**:**

s **=** разборИнструкцииДалее**();**

**break;**

**case** T**.**Всё**:**

s **=** разборИнструкцииВсё**();**

**break;**

**case** T**.**Итог**:**

s **=** разборИнструкцииИтог**();**

**break;**

**case** T**.**Переход**:**

s **=** разборИнструкцииПереход**();**

**break;**

**case** T**.**Для**:**

s **=** разборИнструкцииДля**();**

**break;**

**case** T**.**Синхронизованный**:**

s **=** разборИнструкцииСинхронно**();**

**break;**

**case** T**.**Пробуй**:**

s **=** разборИнструкцииПробуй**();**

**break;**

**case** T**.**Брось**:**

s **=** разборИнструкцииБрось**();**

**break;**

**case** T**.**Масштаб**:**

**if** **(**возьмиСледщ**()** **!=** T**.**ЛСкобка**)**

**goto** случай\_разборАтрибута**;**

s **=** parseScopeGuardStatement**();**

**break;**

**case** T**.**Волатайл**:**

s **=** разборИнструкцииВолатайл**();**

**break;**

**case** T**.**Асм**:**

s **=** parseAsmBlockStatement**();**

**break;**

**case** T**.**Прагма**:**

s **=** разборИнструкцииПрагма**();**

**break;**

**case** T**.**Смесь**:**

**if** **(**возьмиСледщ**()** **==** T**.**ЛСкобка**)**

**goto** случай\_разборИнструкцииВыражения**;** // Parse as выражение.

s **=** парсируйМиксин**!(**ИнструкцияСмесь**)();**

**break;**

**case** T**.**Статический**:**

**switch** **(**возьмиСледщ**())**

**{**

**case** T**.**Если**:**

s **=** разборИнструкцииСтатичЕсли**();**

**break;**

**case** T**.**Подтвердить**:**

s **=** разборИнструкцииСтатичПровер**();**

**break;**

**default:**

**goto** случай\_разборАтрибута**;**

**}**

**break;**

**case** T**.**Отладка**:**

s **=** разборИнструкцииОтладка**();**

**break;**

**case** T**.**Версия**:**

s **=** разборИнструкцииВерсия**();**

**break;**

// DeclDef

**case** T**.**Алиас**,** T**.**Типдеф**:**

d **=** разборДефиницииДекларации**();**

**goto** LreturnDeclarationStatement**;**

**case** T**.**Перечень**:**

**version(**D2**)**

**{**

**if** **(**манифестПеречня\_ли**())**

**goto** случай\_разборАтрибута**;**

**}**

d **=** разборДекларацииПеречня**();**

**goto** LreturnDeclarationStatement**;**

**case** T**.**Класс**:**

d **=** разборДекларацииКласса**();**

**goto** LreturnDeclarationStatement**;**

**case** T**.**Интерфейс**:**

d **=** разборДекларацииИнтерфейса**();**

**goto** LreturnDeclarationStatement**;**

**case** T**.**Структура**,** T**.**Союз**:**

d **=** разборДекларацииСтруктурыИлиСоюза**();**

// goto LreturnDeclarationStatement;

LreturnDeclarationStatement**:**

установи**(**d**,** начало**);**

s **=** **new** ИнструкцияДекларация**(**d**);**

**break;**

**case** T**.**ЛФСкобка**:**

s **=** разборИнструкцииМасштаб**();**

**break;**

**case** T**.**ТочкаЗапятая**:**

далее**();**

s **=** **new** ПустаяИнструкция**();**

**break;**

// Parse an ИнструкцияВыражение:

// Токены that старт a PrimaryExpression.

// case T.Идентификатор, T.Точка, T.Типа:

**case** T**.**Этот**:**

**case** T**.**Супер**:**

**case** T**.**Нуль**:**

**case** T**.**Истина**,** T**.**Ложь**:**

// case T.Доллар:

**case** T**.**Цел32**,** T**.**Цел64**,** T**.**Бцел32**,** T**.**Бцел64**:**

**case** T**.**Плав32**,** T**.**Плав64**,** T**.**Плав80**,**

T**.**Мнимое32**,** T**.**Мнимое64**,** T**.**Мнимое80**:**

**case** T**.**СимЛитерал**:**

**case** T**.**Ткст**:**

**case** T**.**ЛКвСкобка**:**

// case T.ЛФСкобка:

**case** T**.**Функция**,** T**.**Делегат**:**

**case** T**.**Подтвердить**:**

// case T.Смесь:

**case** T**.**Импорт**:**

**case** T**.**Идтипа**:**

**case** T**.**Является**:**

**case** T**.**ЛСкобка**:**

**case** T**.**Трэтс**:** // D2.0

// Токены that can старт a УнарноеВыражение:

**case** T**.**ИБинарное**,** T**.**ПлюсПлюс**,** T**.**МинусМинус**,** T**.**Умножь**,** T**.**Минус**,**

T**.**Плюс**,** T**.**Не**,** T**.**Тильда**,** T**.**Нов**,** T**.**Удалить**,** T**.**Каст**:**

случай\_разборИнструкцииВыражения**:**

s **=** **new** ИнструкцияВыражение**(**разборВыражения**());**

требуется**(**T**.**ТочкаЗапятая**);**

**break;**

**default:**

**if** **(**сема**.**спецСема\_ли**)**

**goto** случай\_разборИнструкцииВыражения**;**

**if** **(**сема**.**вид **!=** T**.**Доллар**)**

// Подтвердить that this isn't a valid выражение.

**assert(delegate** бул**(){**

бул успех**;**

**auto** выражение **=** пробуй\_**(&**разборВыражения**,** успех**);**

**return** успех**;**

**}()** **==** нет**,** "Валидное выражение не ожидалось."

**);**

// Report ошибка: it's an illegal statement.

s **=** **new** НелегальнаяИнструкция**();**

// Skip в следщ valid сема.

**do**

далее**();**

**while** **(!**сема**.**началоИнстр\_ли **&&**

сема**.**вид **!=** T**.**ПФСкобка **&&**

сема**.**вид **!=** T**.**КФ**)**

**auto** текст **=** Сема**.**textSpan**(**начало**,** **this.**предыдущСема**);**

ошибка**(**начало**,** сооб**.**НелегальнаяИнструкция**,** текст**);**

**}**

**assert(**s **!is** **null);**

установи**(**s**,** начало**);**

**return** s**;**

**}**

/// $(PRE

/// Parses a ИнструкцияМасштаб.

/// ИнструкцияМасштаб :=

/// NoScopeStatement

/// )

Инструкция разборИнструкцииМасштаб**()**

**{**

**return** **new** ИнструкцияМасштаб**(**разборИнструкцииБезМасштаба**());**

**}**

/// $(PRE

/// NoScopeStatement :=

/// NonEmptyStatement

/// BlockStatement

/// BlockStatement :=

/// { }

/// { StatementList }

/// )

Инструкция разборИнструкцииБезМасштаба**()**

**{**

**auto** начало **=** сема**;**

Инструкция s**;**

**if** **(**проверено**(**T**.**ЛФСкобка**))**

**{**

**auto** ss **=** **new** СложнаяИнструкция**();**

**while** **(**сема**.**вид **!=** T**.**ПФСкобка **&&** сема**.**вид **!=** T**.**КФ**)**

ss **~=** разборИнструкции**();**

требуетсяЗакрыв**(**T**.**ПФСкобка**,** начало**);**

s **=** установи**(**ss**,** начало**);**

**}**

**else** **if** **(**сема**.**вид **==** T**.**ТочкаЗапятая**)**

**{**

ошибка**(**сема**,** сооб**.**ОжидаласьНеПустаяИнструкция**);**

далее**();**

s **=** установи**(new** ПустаяИнструкция**(),** начало**);**

**}**

**else**

s **=** разборИнструкции**();**

**return** s**;**

**}**

/// $(PRE

/// NoScopeOrEmptyStatement :=

/// ;

/// NoScopeStatement

/// )

Инструкция разборИнструкцииБезМасштабаИлиПустое**()**

**{**

**if** **(**проверено**(**T**.**ТочкаЗапятая**))**

**return** установи**(new** ПустаяИнструкция**(),** **this.**предыдущСема**);**

**else**

**return** разборИнструкцииБезМасштаба**();**

**}**

Инструкция разборИнструкцииАтрибута**()**

**{**

КлассХранения кхр**,** stc\_tmp**;**

ТипКомпоновки предш\_типКомпоновки**;**

Декларация разбор**()** // Nested function.

**{**

**auto** начало **=** сема**;**

Декларация декл**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Экстерн**:**

**if** **(**возьмиСледщ**()** **!=** T**.**ЛСкобка**)**

**{**

stc\_tmp **=** КлассХранения**.**Экстерн**;**

**goto** Lcommon**;**

**}**

далее**();**

**auto** типКомпоновки **=** разборТипаКомпоновки**();**

проверьТипКомпоновки**(**предш\_типКомпоновки**,** типКомпоновки**,** начало**);**

декл **=** **new** ДекларацияКомпоновки**(**типКомпоновки**,** разбор**());**

**break;**

**case** T**.**Статический**:**

stc\_tmp **=** КлассХранения**.**Статический**;**

**goto** Lcommon**;**

**case** T**.**Окончательный**:**

stc\_tmp **=** КлассХранения**.**Окончательный**;**

**goto** Lcommon**;**

**case** T**.**Конст**:**

**version(**D2**)**

**{**

**if** **(**возьмиСледщ**()** **==** T**.**ЛСкобка**)**

**goto** случай\_Декларация**;**

**}**

stc\_tmp **=** КлассХранения**.**Конст**;**

**goto** Lcommon**;**

**version(**D2**)**

**{**

**case** T**.**Инвариант**:** // D 2.0

**if** **(**возьмиСледщ**()** **==** T**.**ЛСкобка**)**

**goto** случай\_Декларация**;**

stc\_tmp **=** КлассХранения**.**Инвариант**;**

**goto** Lcommon**;**

**case** T**.**Перечень**:** // D 2.0

**if** **(!**манифестПеречня\_ли**())**

**{** // A normal enum declaration.

декл **=** разборДекларацииПеречня**();**

// NB: this must be similar в the код at the конец of

// разборДефиницииДекларации().

декл**.**установиЗащиту**(this.**защита**);**

декл**.**установиКлассХранения**(**кхр**);**

установи**(**декл**,** начало**);**

**return** декл**;**

**}**

// enum as КлассХранения.

stc\_tmp **=** КлассХранения**.**Манифест**;**

**goto** Lcommon**;**

**}**

**case** T**.**Авто**:**

stc\_tmp **=** КлассХранения**.**Авто**;**

**goto** Lcommon**;**

**case** T**.**Масштаб**:**

stc\_tmp **=** КлассХранения**.**Масштаб**;**

**goto** Lcommon**;**

Lcommon**:**

// Issue ошибка if redundant.

**if** **(**кхр **&** stc\_tmp**)**

ошибка2**(**ИДС**.**ПовторяющийсяКлассХранения**,** сема**);**

**else**

кхр **|=** stc\_tmp**;**

далее**();**

декл **=** **new** ДекларацияКлассаХранения**(**stc\_tmp**,** разбор**());**

**break;**

**case** T**.**Класс**,** T**.**Интерфейс**,** T**.**Структура**,** T**.**Союз**,** T**.**Алиас**,** T**.**Типдеф**:**

декл **=** разборДефиницииДекларации**();**

декл**.**установиЗащиту**(**Защита**.**Нет**);**

декл**.**установиКлассХранения**(**КлассХранения**.**Нет**);**

**return** декл**;**

**default:**

случай\_Декларация**:**

**return** разборПеременнойИлиФункции**(**кхр**,** Защита**.**Нет**,** предш\_типКомпоновки**,** да**);**

**}**

**return** установи**(**декл**,** начало**);**

**}**

**return** **new** ИнструкцияДекларация**(**разбор**());**

**}**

Инструкция разборИнструкцииЕсли**()**

**{**

пропусти**(**T**.**Если**);**

Инструкция переменная**;**

Выражение условие**;**

Инструкция телоЕсли**,** телоИначе**;**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

Идентификатор**\*** идент**;**

**auto** начало **=** сема**;** // При старт of ДекларацияАвто or normal Декларация.

// auto Идентификатор = Выражение

**if** **(**проверено**(**T**.**Авто**))**

**{**

идент **=** требуетсяИдентификатор**(**сооб**.**ОжидалосьНазваниеПеременной**);**

требуется**(**T**.**Присвоить**);**

**auto** иниц **=** разборВыражения**();**

**auto** v **=** **new** ДекларацияПеременных**(null,** **[**идент**],** **[**иниц**]);**

установи**(**v**,** начало**.**следщНепроб**);**

**auto** d **=** **new** ДекларацияКлассаХранения**(**КлассХранения**.**Авто**,** v**);**

установи**(**d**,** начало**);**

переменная **=** **new** ИнструкцияДекларация**(**d**);**

установи**(**переменная**,** начало**);**

**}**

**else**

**{** // Declarator = Выражение

Тип parseDeclaratorAssign**()**

**{**

**auto** тип **=** разборДекларатора**(**идент**);**

требуется**(**T**.**Присвоить**);**

**return** тип**;**

**}**

бул успех**;**

**auto** тип **=** пробуй\_**(&**parseDeclaratorAssign**,** успех**);**

**if** **(**успех**)**

**{**

**auto** иниц **=** разборВыражения**();**

**auto** v **=** **new** ДекларацияПеременных**(**тип**,** **[**идент**],** **[**иниц**]);**

установи**(**v**,** начало**);**

переменная **=** **new** ИнструкцияДекларация**(**v**);**

установи**(**переменная**,** начало**);**

**}**

**else**

условие **=** разборВыражения**();**

**}**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

телоЕсли **=** разборИнструкцииМасштаб**();**

**if** **(**проверено**(**T**.**Иначе**))**

телоИначе **=** разборИнструкцииМасштаб**();**

**return** **new** ИнструкцияЕсли**(**переменная**,** условие**,** телоЕсли**,** телоИначе**);**

**}**

Инструкция разборИнструкцииПока**()**

**{**

пропусти**(**T**.**Пока**);**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

**auto** условие **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

**return** **new** ИнструкцияПока**(**условие**,** разборИнструкцииМасштаб**());**

**}**

Инструкция разборИнструкцииДелайПока**()**

**{**

пропусти**(**T**.**Делай**);**

**auto** телоДелай **=** разборИнструкцииМасштаб**();**

требуется**(**T**.**Пока**);**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

**auto** условие **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

**return** **new** ИнструкцияДелайПока**(**условие**,** телоДелай**);**

**}**

Инструкция разборИнструкцииПри**()**

**{**

пропусти**(**T**.**При**);**

Инструкция иниц**,** телоПри**;**

Выражение условие**,** инкремент**;**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

**if** **(!**проверено**(**T**.**ТочкаЗапятая**))**

иниц **=** разборИнструкцииБезМасштаба**();**

**if** **(**сема**.**вид **!=** T**.**ТочкаЗапятая**)**

условие **=** разборВыражения**();**

требуется**(**T**.**ТочкаЗапятая**);**

**if** **(**сема**.**вид **!=** T**.**ПСкобка**)**

инкремент **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

телоПри **=** разборИнструкцииМасштаб**();**

**return** **new** ИнструкцияПри**(**иниц**,** условие**,** инкремент**,** телоПри**);**

**}**

Инструкция разборИнструкцииДлявсех**()**

**{**

**assert(**сема**.**вид **==** T**.**Длявсех **||** сема**.**вид **==** T**.**Длявсех\_реверс**);**

TOK лекс **=** сема**.**вид**;**

далее**();**

**auto** парамы **=** **new** Параметры**;**

Выражение в**;** // Агрегат or LwrExpression

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

**auto** paramsBegin **=** сема**;**

**do**

**{**

**auto** paramBegin **=** сема**;**

КлассХранения кхр**;**

Тип тип**;**

Идентификатор**\*** идент**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Реф**,** T**.**Вховых**:**

кхр **=** КлассХранения**.**Реф**;**

далее**();**

// fall through

**case** T**.**Идентификатор**:**

**auto** следщ **=** возьмиСледщ**();**

**if** **(**следщ **==** T**.**Запятая **||** следщ **==** T**.**ТочкаЗапятая **||** следщ **==** T**.**ПСкобка**)**

**{**

идент **=** требуетсяИдентификатор**(**сооб**.**ОжидалосьНазваниеПеременной**);**

**break;**

**}**

// fall through

**default:**

тип **=** разборДекларатора**(**идент**);**

**}**

парамы **~=** установи**(new** Параметр**(**кхр**,** тип**,** идент**,** **null),** paramBegin**);**

**}** **while** **(**проверено**(**T**.**Запятая**))**

установи**(**парамы**,** paramsBegin**);**

требуется**(**T**.**ТочкаЗапятая**);**

в **=** разборВыражения**();**

**version(**D2**)**

**{** //Длявсех (ForeachType; LwrExpression .. UprExpression ) ИнструкцияМасштаб

**if** **(**проверено**(**T**.**Срез**))**

**{**

// if (парамы.length != 1)

// ошибка(ИДС.XYZ); // TODO: issue ошибка сооб

**auto** верхний **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

**auto** телоПри **=** разборИнструкцииМасштаб**();**

**return** **new** ИнструкцияДиапазонСКаждым**(**лекс**,** парамы**,** в**,** верхний**,** телоПри**);**

**}**

**}**

// Длявсех (ForeachTypeList; Агрегат) ИнструкцияМасштаб

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

**auto** телоПри **=** разборИнструкцииМасштаб**();**

**return** **new** ИнструкцияСКаждым**(**лекс**,** парамы**,** в**,** телоПри**);**

**}**

Инструкция разборИнструкцииЩит**()**

**{**

пропусти**(**T**.**Щит**);**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

**auto** условие **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

**auto** телоЩит **=** разборИнструкцииМасштаб**();**

**return** **new** ИнструкцияЩит**(**условие**,** телоЩит**);**

**}**

/// Helper function for parsing the body of a default or case statement.

Инструкция parseCaseOrDefaultBody**()**

**{**

// Этот function is similar в разборИнструкцииБезМасштаба()

**auto** начало **=** сема**;**

**auto** s **=** **new** СложнаяИнструкция**();**

**while** **(**сема**.**вид **!=** T**.**Реле **&&**

сема**.**вид **!=** T**.**Дефолт **&&**

сема**.**вид **!=** T**.**ПФСкобка **&&**

сема**.**вид **!=** T**.**КФ**)**

s **~=** разборИнструкции**();**

установи**(**s**,** начало**);**

**return** установи**(new** ИнструкцияМасштаб**(**s**),** начало**);**

**}**

Инструкция разборИнструкцииРеле**()**

**{**

пропусти**(**T**.**Реле**);**

**auto** значения **=** разборСпискаВыражений**();**

требуется**(**T**.**Двоеточие**);**

**auto** телоРеле **=** parseCaseOrDefaultBody**();**

**return** **new** ИнструкцияРеле**(**значения**,** телоРеле**);**

**}**

Инструкция разборИнструкцииДефолт**()**

**{**

пропусти**(**T**.**Дефолт**);**

требуется**(**T**.**Двоеточие**);**

**auto** телоДефолта **=** parseCaseOrDefaultBody**();**

**return** **new** ИнструкцияДефолт**(**телоДефолта**);**

**}**

Инструкция разборИнструкцииДалее**()**

**{**

пропусти**(**T**.**Далее**);**

**auto** идент **=** дополнительныйИдентификатор**();**

требуется**(**T**.**ТочкаЗапятая**);**

**return** **new** ИнструкцияДалее**(**идент**);**

**}**

Инструкция разборИнструкцииВсё**()**

**{**

пропусти**(**T**.**Всё**);**

**auto** идент **=** дополнительныйИдентификатор**();**

требуется**(**T**.**ТочкаЗапятая**);**

**return** **new** ИнструкцияВсё**(**идент**);**

**}**

Инструкция разборИнструкцииИтог**()**

**{**

пропусти**(**T**.**Итог**);**

Выражение выр**;**

**if** **(**сема**.**вид **!=** T**.**ТочкаЗапятая**)**

выр **=** разборВыражения**();**

требуется**(**T**.**ТочкаЗапятая**);**

**return** **new** ИнструкцияИтог**(**выр**);**

**}**

Инструкция разборИнструкцииПереход**()**

**{**

пропусти**(**T**.**Переход**);**

Идентификатор**\*** идент**;**

Выражение вырРеле**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Реле**:**

идент **=** сема**.**идент**;**

далее**();**

**if** **(**сема**.**вид **==** T**.**ТочкаЗапятая**)**

**break;**

вырРеле **=** разборВыражения**();**

**break;**

**case** T**.**Дефолт**:**

идент **=** сема**.**идент**;**

далее**();**

**break;**

**default:**

идент **=** требуетсяИдентификатор**(**сооб**.**ОжидалсяИдентификатор**);**

**}**

требуется**(**T**.**ТочкаЗапятая**);**

**return** **new** ИнструкцияПереход**(**идент**,** вырРеле**);**

**}**

Инструкция разборИнструкцииДля**()**

**{**

пропусти**(**T**.**Для**);**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

**auto** выр **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

**return** **new** ИнструкцияДля**(**выр**,** разборИнструкцииМасштаб**());**

**}**

Инструкция разборИнструкцииСинхронно**()**

**{**

пропусти**(**T**.**Синхронизованный**);**

Выражение выр**;**

**if** **(**проверено**(**T**.**ЛСкобка**))**

**{**

**auto** leftParen **=** **this.**предыдущСема**;**

выр **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

**}**

**return** **new** ИнструкцияСинхр**(**выр**,** разборИнструкцииМасштаб**());**

**}**

Инструкция разборИнструкцииПробуй**()**

**{**

**auto** начало **=** сема**;**

пропусти**(**T**.**Пробуй**);**

**auto** телоПробуй **=** разборИнструкцииМасштаб**();**

ИнструкцияЛови**[]** телаЛови**;**

ИнструкцияИтожь finBody**;**

**while** **(**проверено**(**T**.**Кэтч**))**

**{**

Параметр парам**;**

**if** **(**проверено**(**T**.**ЛСкобка**))**

**{**

**auto** begin2 **=** сема**;**

Идентификатор**\*** идент**;**

**auto** тип **=** разборДекларатора**(**идент**,** да**);**

парам **=** **new** Параметр**(**КлассХранения**.**Нет**,** тип**,** идент**,** **null);**

установи**(**парам**,** begin2**);**

требуется**(**T**.**ПСкобка**);**

**}**

телаЛови **~=** установи**(new** ИнструкцияЛови**(**парам**,** разборИнструкцииБезМасштаба**()),** начало**);**

**if** **(**парам **is** **null)**

**break;** // Этот is a LastCatch

начало **=** сема**;**

**}**

**if** **(**проверено**(**T**.**Finally**))**

finBody **=** установи**(new** ИнструкцияИтожь**(**разборИнструкцииБезМасштаба**()),** предыдущСема**);**

**if** **(**телаЛови**.**length **==** 0 **&&** finBody **is** **null)**

**assert(**начало**.**вид **==** T**.**Пробуй**),** ошибка**(**начало**,** сооб**.**НеДостаетCatchИлиFinally**);**

**return** **new** ИнструкцияПробуй**(**телоПробуй**,** телаЛови**,** finBody**);**

**}**

Инструкция разборИнструкцииБрось**()**

**{**

пропусти**(**T**.**Брось**);**

**auto** выр **=** разборВыражения**();**

требуется**(**T**.**ТочкаЗапятая**);**

**return** **new** ИнструкцияБрось**(**выр**);**

**}**

Инструкция parseScopeGuardStatement**()**

**{**

пропусти**(**T**.**Масштаб**);**

пропусти**(**T**.**ЛСкобка**);**

**auto** условие **=** требуетсяИдентификатор**(**сооб**.**ОжидалсяИдентификаторМасштаба**);**

**if** **(**условие**)**

**switch** **(**условие**.**видИд**)**

**{**

**case** ВИД**.**выход**,** ВИД**.**успех**,** ВИД**.**сбой**:**

**break;**

**default:**

ошибка2**(**сооб**.**НеверныйИдентификаторМасштаба**,** **this.**предыдущСема**);**

**}**

требуется**(**T**.**ПСкобка**);**

Инструкция телоМасштаба**;**

**if** **(**сема**.**вид **==** T**.**ЛФСкобка**)**

телоМасштаба **=** разборИнструкцииМасштаб**();**

**else**

телоМасштаба **=** разборИнструкцииБезМасштаба**();**

**return** **new** ИнструкцияСтражМасштаба**(**условие**,** телоМасштаба**);**

**}**

Инструкция разборИнструкцииВолатайл**()**

**{**

пропусти**(**T**.**Волатайл**);**

Инструкция телоЛетучего**;**

**if** **(**сема**.**вид **==** T**.**ТочкаЗапятая**)**

далее**();**

**else** **if** **(**сема**.**вид **==** T**.**ЛФСкобка**)**

телоЛетучего **=** разборИнструкцииМасштаб**();**

**else**

телоЛетучего **=** разборИнструкции**();**

**return** **new** ИнструкцияЛетучее**(**телоЛетучего**);**

**}**

Инструкция разборИнструкцииПрагма**()**

**{**

пропусти**(**T**.**Прагма**);**

Идентификатор**\*** идент**;**

Выражение**[]** арги**;**

Инструкция телоПрагмы**;**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

идент **=** требуетсяИдентификатор**(**сооб**.**ОжидалсяИдентификаторПрагмы**);**

**if** **(**проверено**(**T**.**Запятая**))**

арги **=** разборСпискаВыражений**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

телоПрагмы **=** разборИнструкцииБезМасштабаИлиПустое**();**

**return** **new** ИнструкцияПрагма**(**идент**,** арги**,** телоПрагмы**);**

**}**

Инструкция разборИнструкцииСтатичЕсли**()**

**{**

пропусти**(**T**.**Статический**);**

пропусти**(**T**.**Если**);**

Выражение условие**;**

Инструкция телоЕсли**,** телоИначе**;**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

условие **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

телоЕсли **=** разборИнструкцииБезМасштаба**();**

**if** **(**проверено**(**T**.**Иначе**))**

телоИначе **=** разборИнструкцииБезМасштаба**();**

**return** **new** ИнструкцияСтатическоеЕсли**(**условие**,** телоЕсли**,** телоИначе**);**

**}**

Инструкция разборИнструкцииСтатичПровер**()**

**{**

пропусти**(**T**.**Статический**);**

пропусти**(**T**.**Подтвердить**);**

Выражение условие**,** сообщение**;**

требуется**(**T**.**ЛСкобка**);**

условие **=** разборВыраженияПрисвой**();** // Condition.

**if** **(**проверено**(**T**.**Запятая**))**

сообщение **=** разборВыраженияПрисвой**();** // Ошибка сообщение.

требуется**(**T**.**ПСкобка**);**

требуется**(**T**.**ТочкаЗапятая**);**

**return** **new** ИнструкцияСтатическоеПодтверди**(**условие**,** сообщение**);**

**}**

Инструкция разборИнструкцииОтладка**()**

**{**

пропусти**(**T**.**Отладка**);**

Сема**\*** услов**;**

Инструкция телоОтладки**,** телоИначе**;**

// ( Condition )

**if** **(**проверено**(**T**.**ЛСкобка**))**

**{**

услов **=** разборИдентИлиЦел**();**

требуется**(**T**.**ПСкобка**);**

**}**

// debug Инструкция

// debug ( Condition ) Инструкция

телоОтладки **=** разборИнструкцииБезМасштаба**();**

// else Инструкция

**if** **(**проверено**(**T**.**Иначе**))**

телоИначе **=** разборИнструкцииБезМасштаба**();**

**return** **new** ИнструкцияОтладка**(**услов**,** телоОтладки**,** телоИначе**);**

**}**

Инструкция разборИнструкцииВерсия**()**

**{**

пропусти**(**T**.**Версия**);**

Сема**\*** услов**;**

Инструкция телоВерсии**,** телоИначе**;**

// ( Condition )

требуется**(**T**.**ЛСкобка**);**

услов **=** разборУсловияВерсии**();**

требуется**(**T**.**ПСкобка**);**

// version ( Condition ) Инструкция

телоВерсии **=** разборИнструкцииБезМасштаба**();**

// else Инструкция

**if** **(**проверено**(**T**.**Иначе**))**

телоИначе **=** разборИнструкцииБезМасштаба**();**

**return** **new** ИнструкцияВерсия**(**услов**,** телоВерсии**,** телоИначе**);**

**}**

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

| Assembler parsing methods |

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

/// Parses an ИнструкцияБлокАсм.

Инструкция parseAsmBlockStatement**()**

**{**

пропусти**(**T**.**Асм**);**

**auto** леваяФСкобка **=** сема**;**

требуется**(**T**.**ЛФСкобка**);**

**auto** ss **=** **new** СложнаяИнструкция**;**

**while** **(**сема**.**вид **!=** T**.**ПФСкобка **&&** сема**.**вид **!=** T**.**КФ**)**

ss **~=** parseAsmStatement**();**

требуетсяЗакрыв**(**T**.**ПФСкобка**,** леваяФСкобка**);**

**return** **new** ИнструкцияБлокАсм**(**ss**);**

**}**

Инструкция parseAsmStatement**()**

**{**

**auto** начало **=** сема**;**

Инструкция s**;**

Идентификатор**\*** идент**;**

**switch** **(**сема**.**вид**)**

**{**

// Keywords that are valid opcodes.

**case** T**.**Вхо**,** T**.**Цел**,** T**.**Вых**:**

идент **=** сема**.**идент**;**

далее**();**

**goto** LOpcode**;**

**case** T**.**Идентификатор**:**

идент **=** сема**.**идент**;**

далее**();**

**if** **(**проверено**(**T**.**Двоеточие**))**

**{** // Идентификатор : ИнструкцияАсм

s **=** **new** ИнструкцияСМеткой**(**идент**,** parseAsmStatement**());**

**break;**

**}**

LOpcode**:**

// Opcode ;

// Opcode Operands ;

// Opcode

// Идентификатор

Выражение**[]** es**;**

**if** **(**сема**.**вид **!=** T**.**ТочкаЗапятая**)**

**do**

es **~=** parseAsmExpression**();**

**while** **(**проверено**(**T**.**Запятая**))**

требуется**(**T**.**ТочкаЗапятая**);**

s **=** **new** ИнструкцияАсм**(**идент**,** es**);**

**break;**

**case** T**.**Расклад**:**

// align Integer;

далее**();**

цел число **=** **-**1**;**

**if** **(**сема**.**вид **==** T**.**Цел32**)**

**(**число **=** сема**.**цел\_**),** пропусти**(**T**.**Цел32**);**

**else**

ошибка2**(**сооб**.**ExpectedIntegerAfterAlign**,** сема**);**

требуется**(**T**.**ТочкаЗапятая**);**

s **=** **new** ИнструкцияАсмРасклад**(**число**);**

**break;**

**case** T**.**ТочкаЗапятая**:**

s **=** **new** ПустаяИнструкция**();**

далее**();**

**break;**

**default:**

s **=** **new** ИнструкцияНелегальныйАсм**();**

// Skip в следщ valid сема.

**do**

далее**();**

**while** **(!**сема**.**началоАсмИнстр\_ли **&&**

сема**.**вид **!=** T**.**ПФСкобка **&&**

сема**.**вид **!=** T**.**КФ**)**

**auto** текст **=** Сема**.**textSpan**(**начало**,** **this.**предыдущСема**);**

ошибка**(**начало**,** сооб**.**ИнструкцияНелегальныйАсм**,** текст**);**

**}**

установи**(**s**,** начало**);**

**return** s**;**

**}**

Выражение parseAsmExpression**()**

**{**

**auto** начало **=** сема**;**

**auto** в **=** parseAsmOrOrExpression**();**

**if** **(**проверено**(**T**.**Вопрос**))**

**{**

**auto** лекс **=** **this.**предыдущСема**;**

**auto** iftrue **=** parseAsmExpression**();**

требуется**(**T**.**Двоеточие**);**

**auto** iffalse **=** parseAsmExpression**();**

в **=** **new** ВыражениеУсловия**(**в**,** iftrue**,** iffalse**,** лекс**);**

установи**(**в**,** начало**);**

**}**

// TODO: create AsmExpression that contains в?

**return** в**;**

**}**

Выражение parseAsmOrOrExpression**()**

**{**

**alias** parseAsmAndAndExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**ИлиЛогическое**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

в **=** **new** ВыражениеИлиИли**(**в**,** разборСледующего**(),** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseAsmAndAndExpression**()**

**{**

**alias** parseAsmOrExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**ИЛогическое**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

в **=** **new** ВыражениеИИ**(**в**,** разборСледующего**(),** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseAsmOrExpression**()**

**{**

**alias** parseAsmXorExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**ИлиБинарное**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

в **=** **new** ВыражениеИли**(**в**,** разборСледующего**(),** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseAsmXorExpression**()**

**{**

**alias** parseAsmAndExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**ИИли**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

в **=** **new** ВыражениеИИли**(**в**,** разборСледующего**(),** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseAsmAndExpression**()**

**{**

**alias** parseAsmCmpExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**ИБинарное**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

в **=** **new** ВыражениеИ**(**в**,** разборСледующего**(),** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseAsmCmpExpression**()**

**{**

**alias** parseAsmShiftExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**auto** operator **=** сема**;**

**switch** **(**operator**.**вид**)**

**{**

**case** T**.**Равно**,** T**.**НеРавно**:**

далее**();**

в **=** **new** ВыражениеРавно**(**в**,** разборСледующего**(),** operator**);**

**break;**

**case** T**.**МеньшеРавно**,** T**.**Меньше**,** T**.**БольшеРавно**,** T**.**Больше**:**

далее**();**

в **=** **new** ВыражениеОтнош**(**в**,** разборСледующего**(),** operator**);**

**break;**

**default:**

**return** в**;**

**}**

установи**(**в**,** начало**);**

**return** в**;**

**}**

Выражение parseAsmShiftExpression**()**

**{**

**alias** parseAsmAddExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**1**)**

**{**

**auto** operator **=** сема**;**

**switch** **(**operator**.**вид**)**

**{**

**case** T**.**ЛСдвиг**:** далее**();** в **=** **new** ВыражениеЛСдвиг**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**ПСдвиг**:** далее**();** в **=** **new** ВыражениеПСдвиг**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**URShift**:** далее**();** в **=** **new** ВыражениеБПСдвиг**(**в**,** разборСледующего**(),** operator**);** **break;**

**default:**

**return** в**;**

**}**

установи**(**в**,** начало**);**

**}**

**assert(**0**);**

**}**

Выражение parseAsmAddExpression**()**

**{**

**alias** parseAsmMulExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**1**)**

**{**

**auto** operator **=** сема**;**

**switch** **(**operator**.**вид**)**

**{**

**case** T**.**Плюс**:** далее**();** в **=** **new** ВыражениеПлюс**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**Минус**:** далее**();** в **=** **new** ВыражениеМинус**(**в**,** разборСледующего**(),** operator**);** **break;**

// Не allowed in asm

//case T.Тильда: далее(); в = new ВыражениеСоедини(в, разборСледующего(), operator); break;

**default:**

**return** в**;**

**}**

установи**(**в**,** начало**);**

**}**

**assert(**0**);**

**}**

Выражение parseAsmMulExpression**()**

**{**

**alias** parseAsmPostExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**1**)**

**{**

**auto** operator **=** сема**;**

**switch** **(**operator**.**вид**)**

**{**

**case** T**.**Умножь**:** далее**();** в **=** **new** ВыражениеУмножь**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**Деление**:** далее**();** в **=** **new** ВыражениеДели**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**Модуль**:** далее**();** в **=** **new** ВыражениеМод**(**в**,** разборСледующего**(),** operator**);** **break;**

**default:**

**return** в**;**

**}**

установи**(**в**,** начало**);**

**}**

**assert(**0**);**

**}**

Выражение parseAsmPostExpression**()**

**{**

**auto** начало **=** сема**;**

**auto** в **=** parseAsmUnaryExpression**();**

**while** **(**проверено**(**T**.**ЛКвСкобка**))**

**{**

**auto** leftBracket **=** **this.**предыдущСема**;**

в **=** **new** ВыражениеАсмПослеСкобки**(**в**,** parseAsmExpression**());**

требуетсяЗакрыв**(**T**.**ПКвСкобка**,** leftBracket**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseAsmUnaryExpression**()**

**{**

**auto** начало **=** сема**;**

Выражение в**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Байт**,** T**.**Крат**,** T**.**Цел**,**

T**.**Плав**,** T**.**Дво**,** T**.**Реал**:**

**goto** LAsmTypePrefix**;**

**case** T**.**Идентификатор**:**

**switch** **(**сема**.**идент**.**видИд**)**

**{**

**case** ВИД**.**near**,** ВИД**.**far**,**/\* "байт", "крат", "цел",\*/

ВИД**.**word**,** ВИД**.**dword**,** ВИД**.**qword/\*, "плав", "дво", "реал"\*/**:**

LAsmTypePrefix**:**

далее**();**

**if** **(**сема**.**вид **==** T**.**Идентификатор **&&** сема**.**идент **is** Идент**.**ptr**)**

пропусти**(**T**.**Идентификатор**);**

**else**

ошибка2**(**ИДС**.**НайденоИноеЧемОжидалось**,** "ptr"**,** сема**);**

в **=** **new** ВыражениеТипАсм**(**parseAsmExpression**());**

**break;**

**case** ВИД**.**offset**:**

далее**();**

в **=** **new** ВыражениеСмещениеАсм**(**parseAsmExpression**());**

**break;**

**case** ВИД**.**seg**:**

далее**();**

в **=** **new** ВыражениеСегАсм**(**parseAsmExpression**());**

**break;**

**default:**

**goto** LparseAsmPrimaryExpression**;**

**}**

**break;**

**case** T**.**Минус**:**

**case** T**.**Плюс**:**

далее**();**

в **=** **new** ВыражениеЗнак**(**parseAsmUnaryExpression**());**

**break;**

**case** T**.**Не**:**

далее**();**

в **=** **new** ВыражениеНе**(**parseAsmUnaryExpression**());**

**break;**

**case** T**.**Тильда**:**

далее**();**

в **=** **new** ВыражениеКомп**(**parseAsmUnaryExpression**());**

**break;**

**case** T**.**Точка**:**

далее**();**

в **=** **new** ВыражениеМасштабМодуля**(**разборВыраженияИдентификатора**());**

**while** **(**проверено**(**TOK**.**Точка**))**

**{**

в **=** **new** ВыражениеТочка**(**в**,** разборВыраженияИдентификатора**());**

установи**(**в**,** начало**);**

**}**

**break;**

**default:**

LparseAsmPrimaryExpression**:**

в **=** parseAsmPrimaryExpression**();**

**return** в**;**

**}**

установи**(**в**,** начало**);**

**return** в**;**

**}**

Выражение parseAsmPrimaryExpression**()**

**{**

**auto** начало **=** сема**;**

Выражение в**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Цел32**,** T**.**Цел64**,** T**.**Бцел32**,** T**.**Бцел64**:**

в **=** **new** ЦелВыражение**(**сема**);**

далее**();**

**break;**

**case** T**.**Плав32**,** T**.**Плав64**,** T**.**Плав80**,**

T**.**Мнимое32**,** T**.**Мнимое64**,** T**.**Мнимое80**:**

в **=** **new** ВыражениеРеал**(**сема**);**

далее**();**

**break;**

**case** T**.**Доллар**:**

в **=** **new** ВыражениеДоллар**();**

далее**();**

**break;**

**case** T**.**ЛКвСкобка**:**

// [ AsmExpression ]

**auto** leftBracket **=** сема**;**

далее**();**

в **=** parseAsmExpression**();**

требуетсяЗакрыв**(**T**.**ПКвСкобка**,** leftBracket**);**

в **=** **new** ВыражениеАсмСкобка**(**в**);**

**break;**

**case** T**.**Идентификатор**:**

**auto** регистр **=** сема**.**идент**;**

**switch** **(**регистр**.**видИд**)**

**{**

// \_\_LOCAL\_SIZE

**case** ВИД**.**\_\_LOCAL\_SIZE**:**

далее**();**

в **=** **new** ВыражениеЛокальногоРазмераАсм**();**

**break;**

// Register

**case** ВИД**.**ST**:**

далее**();**

// (1) - (7)

цел число **=** **-**1**;**

**if** **(**проверено**(**T**.**ЛСкобка**))**

**{**

**if** **(**сема**.**вид **==** T**.**Цел32**)**

**(**число **=** сема**.**цел\_**),** пропусти**(**T**.**Цел32**);**

**else**

ожидаемое**(**T**.**Цел32**);**

требуется**(**T**.**ПСкобка**);**

**}**

в **=** **new** ВыражениеАсмРегистр**(**регистр**,** число**);**

**break;**

**case** ВИД**.**FS**:**

далее**();**

// TODO: is the colon-число part optional?

цел число **=** **-**1**;**

**if** **(**проверено**(**T**.**Двоеточие**))**

**{**

// :0, :4, :8

**if** **(**сема**.**вид **==** T**.**Цел32**)**

**(**число **=** сема**.**цел\_**),** пропусти**(**T**.**Цел32**);**

**if** **(**число **!=** 0 **&&** число **!=** 4 **&&** число **!=** 8**)**

ошибка2**(**ИДС**.**НайденоИноеЧемОжидалось**,** "0, 4 or 8"**,** сема**);**

**}**

в **=** **new** ВыражениеАсмРегистр**(**регистр**,** число**);**

**break;**

**case** ВИД**.**AL**,** ВИД**.**AH**,** ВИД**.**AX**,** ВИД**.**EAX**,**

ВИД**.**BL**,** ВИД**.**BH**,** ВИД**.**BX**,** ВИД**.**EBX**,**

ВИД**.**CL**,** ВИД**.**CH**,** ВИД**.**CX**,** ВИД**.**ECX**,**

ВИД**.**DL**,** ВИД**.**DH**,** ВИД**.**DX**,** ВИД**.**EDX**,**

ВИД**.**BP**,** ВИД**.**EBP**,** ВИД**.**SP**,** ВИД**.**ESP**,**

ВИД**.**DI**,** ВИД**.**EDI**,** ВИД**.**SI**,** ВИД**.**ESI**,**

ВИД**.**ES**,** ВИД**.**CS**,** ВИД**.**SS**,** ВИД**.**DS**,** ВИД**.**GS**,**

ВИД**.**CR0**,** ВИД**.**CR2**,** ВИД**.**CR3**,** ВИД**.**CR4**,**

ВИД**.**DR0**,** ВИД**.**DR1**,** ВИД**.**DR2**,** ВИД**.**DR3**,** ВИД**.**DR6**,** ВИД**.**DR7**,**

ВИД**.**TR3**,** ВИД**.**TR4**,** ВИД**.**TR5**,** ВИД**.**TR6**,** ВИД**.**TR7**,**

ВИД**.**MM0**,** ВИД**.**MM1**,** ВИД**.**MM2**,** ВИД**.**MM3**,**

ВИД**.**MM4**,** ВИД**.**MM5**,** ВИД**.**MM6**,** ВИД**.**MM7**,**

ВИД**.**XMM0**,** ВИД**.**XMM1**,** ВИД**.**XMM2**,** ВИД**.**XMM3**,**

ВИД**.**XMM4**,** ВИД**.**XMM5**,** ВИД**.**XMM6**,** ВИД**.**XMM7**:**

далее**();**

в **=** **new** ВыражениеАсмРегистр**(**регистр**);**

**break;**

**default:**

в **=** разборВыраженияИдентификатора**();**

**while** **(**проверено**(**TOK**.**Точка**))**

**{**

в **=** **new** ВыражениеТочка**(**в**,** разборВыраженияИдентификатора**());**

установи**(**в**,** начало**);**

**}**

**}** // конец of switch

**break;**

**default:**

ошибка2**(**ИДС**.**НайденоИноеЧемОжидалось**,** "Выражение"**,** сема**);**

в **=** **new** НелегальноеВыражение**();**

**if** **(!**пробуем**)**

**{** // Insert a dummy сема and don't consume current one.

начало **=** лексер**.**вставьПустуюСемуПеред**(**сема**);**

**this.**предыдущСема **=** начало**;**

**}**

**}**

установи**(**в**,** начало**);**

**return** в**;**

**}**

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

| Выражение parsing methods |

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

/// Parses an Выражение.

Выражение разборВыражения**()**

**{**

**alias** разборВыраженияПрисвой разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**Запятая**)**

**{**

**auto** comma **=** сема**;**

далее**();**

в **=** **new** ВыражениеЗапятая**(**в**,** разборСледующего**(),** comma**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение разборВыраженияПрисвой**()**

**{**

**alias** разборВыраженияПрисвой разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборУсловнВыражения**();**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Присвоить**:**

далее**();** в **=** **new** ВыражениеПрисвой**(**в**,** разборСледующего**());** **break;**

**case** T**.**ЛСдвигПрисвой**:**

далее**();** в **=** **new** ВыражениеПрисвойЛСдвиг**(**в**,** разборСледующего**());** **break;**

**case** T**.**ПСдвигПрисвой**:**

далее**();** в **=** **new** ВыражениеПрисвойПСдвиг**(**в**,** разборСледующего**());** **break;**

**case** T**.**URShiftAssign**:**

далее**();** в **=** **new** ВыражениеПрисвойБПСдвиг**(**в**,** разборСледующего**());** **break;**

**case** T**.**ИлиПрисвой**:**

далее**();** в **=** **new** ВыражениеПрисвойИли**(**в**,** разборСледующего**());** **break;**

**case** T**.**ИПрисвой**:**

далее**();** в **=** **new** ВыражениеПрисвойИ**(**в**,** разборСледующего**());** **break;**

**case** T**.**ПлюсПрисвой**:**

далее**();** в **=** **new** ВыражениеПрисвойПлюс**(**в**,** разборСледующего**());** **break;**

**case** T**.**МинусПрисвой**:**

далее**();** в **=** **new** ВыражениеПрисвойМинус**(**в**,** разборСледующего**());** **break;**

**case** T**.**ДелениеПрисвой**:**

далее**();** в **=** **new** ВыражениеПрисвойДел**(**в**,** разборСледующего**());** **break;**

**case** T**.**УмножьПрисвой**:**

далее**();** в **=** **new** ВыражениеПрисвойУмн**(**в**,** разборСледующего**());** **break;**

**case** T**.**МодульПрисвой**:**

далее**();** в **=** **new** ВыражениеПрисвойМод**(**в**,** разборСледующего**());** **break;**

**case** T**.**ИИлиПрисвой**:**

далее**();** в **=** **new** ВыражениеПрисвойИИли**(**в**,** разборСледующего**());** **break;**

**case** T**.**CatAssign**:**

далее**();** в **=** **new** ВыражениеПрисвойСоед**(**в**,** разборСледующего**());** **break;**

**default:**

**return** в**;**

**}**

установи**(**в**,** начало**);**

**return** в**;**

**}**

Выражение разборУсловнВыражения**()**

**{**

**auto** начало **=** сема**;**

**auto** в **=** parseOrOrExpression**();**

**if** **(**сема**.**вид **==** T**.**Вопрос**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

**auto** iftrue **=** разборВыражения**();**

требуется**(**T**.**Двоеточие**);**

**auto** iffalse **=** разборУсловнВыражения**();**

в **=** **new** ВыражениеУсловия**(**в**,** iftrue**,** iffalse**,** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseOrOrExpression**()**

**{**

**alias** parseAndAndExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**ИлиЛогическое**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

в **=** **new** ВыражениеИлиИли**(**в**,** разборСледующего**(),** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseAndAndExpression**()**

**{**

**alias** parseOrExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**ИЛогическое**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

в **=** **new** ВыражениеИИ**(**в**,** разборСледующего**(),** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseOrExpression**()**

**{**

**alias** parseXorExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**ИлиБинарное**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

в **=** **new** ВыражениеИли**(**в**,** разборСледующего**(),** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseXorExpression**()**

**{**

**alias** parseAndExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**ИИли**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

в **=** **new** ВыражениеИИли**(**в**,** разборСледующего**(),** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseAndExpression**()**

**{**

**alias** parseCmpExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**сема**.**вид **==** T**.**ИБинарное**)**

**{**

**auto** лекс **=** сема**;**

далее**();**

в **=** **new** ВыражениеИ**(**в**,** разборСледующего**(),** лекс**);**

установи**(**в**,** начало**);**

**}**

**return** в**;**

**}**

Выражение parseCmpExpression**()**

**{**

**alias** parseShiftExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** parseShiftExpression**();**

**auto** operator **=** сема**;**

**switch** **(**operator**.**вид**)**

**{**

**case** T**.**Равно**,** T**.**НеРавно**:**

далее**();**

в **=** **new** ВыражениеРавно**(**в**,** разборСледующего**(),** operator**);**

**break;**

**case** T**.**Не**:**

**if** **(**возьмиСледщ**()** **!=** T**.**Является**)**

**break;**

далее**();**

// fall through

**case** T**.**Является**:**

далее**();**

в **=** **new** ВыражениеРавенство**(**в**,** разборСледующего**(),** operator**);**

**break;**

**case** T**.**МеньшеРавно**,** T**.**Меньше**,** T**.**БольшеРавно**,** T**.**Больше**,**

T**.**Unordered**,** T**.**UorE**,** T**.**UorG**,** T**.**UorGorE**,**

T**.**UorL**,** T**.**UorLorE**,** T**.**LorEorG**,** T**.**LorG**:**

далее**();**

в **=** **new** ВыражениеОтнош**(**в**,** разборСледующего**(),** operator**);**

**break;**

**case** T**.**Вхо**:**

далее**();**

в **=** **new** ВыражениеВхо**(**в**,** разборСледующего**(),** operator**);**

**break;**

**default:**

**return** в**;**

**}**

установи**(**в**,** начало**);**

**return** в**;**

**}**

Выражение parseShiftExpression**()**

**{**

**alias** parseAddExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**1**)**

**{**

**auto** operator **=** сема**;**

**switch** **(**operator**.**вид**)**

**{**

**case** T**.**ЛСдвиг**:** далее**();** в **=** **new** ВыражениеЛСдвиг**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**ПСдвиг**:** далее**();** в **=** **new** ВыражениеПСдвиг**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**URShift**:** далее**();** в **=** **new** ВыражениеБПСдвиг**(**в**,** разборСледующего**(),** operator**);** **break;**

**default:**

**return** в**;**

**}**

установи**(**в**,** начало**);**

**}**

**assert(**0**);**

**}**

Выражение parseAddExpression**()**

**{**

**alias** parseMulExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**1**)**

**{**

**auto** operator **=** сема**;**

**switch** **(**operator**.**вид**)**

**{**

**case** T**.**Плюс**:** далее**();** в **=** **new** ВыражениеПлюс**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**Минус**:** далее**();** в **=** **new** ВыражениеМинус**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**Тильда**:** далее**();** в **=** **new** ВыражениеСоедини**(**в**,** разборСледующего**(),** operator**);** **break;**

**default:**

**return** в**;**

**}**

установи**(**в**,** начало**);**

**}**

**assert(**0**);**

**}**

Выражение parseMulExpression**()**

**{**

**alias** parsePostExpression разборСледующего**;**

**auto** начало **=** сема**;**

**auto** в **=** разборСледующего**();**

**while** **(**1**)**

**{**

**auto** operator **=** сема**;**

**switch** **(**operator**.**вид**)**

**{**

**case** T**.**Умножь**:** далее**();** в **=** **new** ВыражениеУмножь**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**Деление**:** далее**();** в **=** **new** ВыражениеДели**(**в**,** разборСледующего**(),** operator**);** **break;**

**case** T**.**Модуль**:** далее**();** в **=** **new** ВыражениеМод**(**в**,** разборСледующего**(),** operator**);** **break;**

**default:**

**return** в**;**

**}**

установи**(**в**,** начало**);**

**}**

**assert(**0**);**

**}**

Выражение parsePostExpression**()**

**{**

**auto** начало **=** сема**;**

**auto** в **=** parseUnaryExpression**();**

**while** **(**1**)**

**{**

**while** **(**проверено**(**T**.**Точка**))**

**{**

в **=** **new** ВыражениеТочка**(**в**,** parseNewOrIdentifierExpression**());**

установи**(**в**,** начало**);**

**}**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**ПлюсПлюс**:**

в **=** **new** ВыражениеПостИнкр**(**в**);**

**break;**

**case** T**.**МинусМинус**:**

в **=** **new** ВыражениеПостДекр**(**в**);**

**break;**

**case** T**.**ЛСкобка**:**

в **=** **new** ВыражениеВызов**(**в**,** разборАргументов**());**

**goto** Lset**;**

**case** T**.**ЛКвСкобка**:**

// разбор Срез- and ВыражениеИндекс

**auto** leftBracket **=** сема**;**

далее**();**

// [] is a ВыражениеСрез

**if** **(**сема**.**вид **==** T**.**ПКвСкобка**)**

**{**

в **=** **new** ВыражениеСрез**(**в**,** **null,** **null);**

**break;**

**}**

Выражение**[]** es **=** **[**разборВыраженияПрисвой**()];**

// [ ВыражениеПрисвой .. ВыражениеПрисвой ]

**if** **(**проверено**(**T**.**Срез**))**

**{**

в **=** **new** ВыражениеСрез**(**в**,** es**[**0**],** разборВыраженияПрисвой**());**

требуетсяЗакрыв**(**T**.**ПКвСкобка**,** leftBracket**);**

**goto** Lset**;**

**}**

// [ ExpressionList ]

**if** **(**проверено**(**T**.**Запятая**))**

es **~=** разборСпискаВыражений**();**

требуетсяЗакрыв**(**T**.**ПКвСкобка**,** leftBracket**);**

в **=** **new** ВыражениеИндекс**(**в**,** es**);**

**goto** Lset**;**

**default:**

**return** в**;**

**}**

далее**();**

Lset**:** // Jumped here в пропусти далее().

установи**(**в**,** начало**);**

**}**

**assert(**0**);**

**}**

Выражение parseUnaryExpression**()**

**{**

**auto** начало **=** сема**;**

Выражение в**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**ИБинарное**:**

далее**();**

в **=** **new** ВыражениеАдрес**(**parseUnaryExpression**());**

**break;**

**case** T**.**ПлюсПлюс**:**

далее**();**

в **=** **new** ВыражениеПреИнкр**(**parseUnaryExpression**());**

**break;**

**case** T**.**МинусМинус**:**

далее**();**

в **=** **new** ВыражениеПреДекр**(**parseUnaryExpression**());**

**break;**

**case** T**.**Умножь**:**

далее**();**

в **=** **new** ВыражениеДереф**(**parseUnaryExpression**());**

**break;**

**case** T**.**Минус**:**

**case** T**.**Плюс**:**

далее**();**

в **=** **new** ВыражениеЗнак**(**parseUnaryExpression**());**

**break;**

**case** T**.**Не**:**

далее**();**

в **=** **new** ВыражениеНе**(**parseUnaryExpression**());**

**break;**

**case** T**.**Тильда**:**

далее**();**

в **=** **new** ВыражениеКомп**(**parseUnaryExpression**());**

**break;**

**case** T**.**Нов**:**

в **=** разборВыраженияНов**();**

**return** в**;**

**case** T**.**Удалить**:**

далее**();**

в **=** **new** ВыражениеУдали**(**parseUnaryExpression**());**

**break;**

**case** T**.**Каст**:**

требуетсяСледующий**(**T**.**ЛСкобка**);**

Тип тип**;**

**switch** **(**сема**.**вид**)**

**{**

**version(**D2**)**

**{**

**auto** begin2 **=** сема**;**

**case** T**.**Конст**:**

тип **=** **new** ТКонст**(null);**

**goto** case\_break**;**

**case** T**.**Инвариант**:**

тип **=** **new** ТИнвариант**(null);**

case\_break**:**

далее**();**

установи**(**тип**,** begin2**);**

**break;**

**}**

**default:**

тип **=** разборТипа**();**

**}**

требуется**(**T**.**ПСкобка**);**

в **=** **new** ВыражениеКаст**(**parseUnaryExpression**(),** тип**);**

**break;**

**case** T**.**ЛСкобка**:**

// ( Тип ) . Идентификатор

Тип parseType\_**()**

**{**

пропусти**(**T**.**ЛСкобка**);**

**auto** тип **=** разборТипа**();**

требуется**(**T**.**ПСкобка**);**

требуется**(**T**.**Точка**);**

**return** тип**;**

**}**

бул успех**;**

**auto** тип **=** пробуй\_**(&**parseType\_**,** успех**);**

**if** **(**успех**)**

**{**

**auto** идент **=** требуетсяИдентификатор**(**сооб**.**ExpectedIdAfterTypeDot**);**

в **=** **new** ВыражениеИдТипаТочка**(**тип**,** идент**);**

**break;**

**}**

**goto** **default;**

**case** T**.**Точка**:**

далее**();**

в **=** **new** ВыражениеМасштабМодуля**(**разборВыраженияИдентификатора**());**

**break;**

**default:**

в **=** parsePrimaryExpression**();**

**return** в**;**

**}**

**assert(**в **!is** **null);**

установи**(**в**,** начало**);**

**return** в**;**

**}**

/// $(PRE

/// ВыражениеИдентификатор :=

/// Идентификатор

/// TemplateInstance

/// TemplateInstance :=

/// Идентификатор !( АргументыШаблона )

/// )

Выражение разборВыраженияИдентификатора**()**

**{**

**auto** начало **=** сема**;**

**auto** идент **=** требуетсяИдентификатор**(**сооб**.**ОжидалсяИдентификатор**);**

Выражение в**;**

// Peek for '(' в avoid matching: ид !is ид

**if** **(**сема**.**вид **==** T**.**Не **&&** возьмиСледщ**()** **==** T**.**ЛСкобка**)**

**{** // Идентификатор !( АргументыШаблона )

пропусти**(**T**.**Не**);**

**auto** шпарамы **=** parseTemplateArguments**();**

в **=** **new** ВыражениеЭкземплярШаблона**(**идент**,** шпарамы**);**

**}**

**else** // Идентификатор

в **=** **new** ВыражениеИдентификатор**(**идент**);**

**return** установи**(**в**,** начало**);**

**}**

Выражение parseNewOrIdentifierExpression**()**

**{**

**return** сема**.**вид **==** T**.**Нов **?** разборВыраженияНов**()** **:** разборВыраженияИдентификатора**();**

**}**

Выражение parsePrimaryExpression**()**

**{**

**auto** начало **=** сема**;**

Выражение в**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Идентификатор**:**

в **=** разборВыраженияИдентификатора**();**

**return** в**;**

**case** T**.**Типа**:**

в **=** **new** ВыражениеТипа**(**parseTypeofType**());**

**break;**

**case** T**.**Этот**:**

далее**();**

в **=** **new** ВыражениеЭтот**();**

**break;**

**case** T**.**Супер**:**

далее**();**

в **=** **new** ВыражениеСупер**();**

**break;**

**case** T**.**Нуль**:**

далее**();**

в **=** **new** ВыражениеНуль**();**

**break;**

**case** T**.**Истина**,** T**.**Ложь**:**

далее**();**

в **=** **new** БулевоВыражение**(**сема**.**вид **==** T**.**Истина**);**

**break;**

**case** T**.**Доллар**:**

далее**();**

в **=** **new** ВыражениеДоллар**();**

**break;**

**case** T**.**Цел32**,** T**.**Цел64**,** T**.**Бцел32**,** T**.**Бцел64**:**

в **=** **new** ЦелВыражение**(**сема**);**

далее**();**

**break;**

**case** T**.**Плав32**,** T**.**Плав64**,** T**.**Плав80**,**

T**.**Мнимое32**,** T**.**Мнимое64**,** T**.**Мнимое80**:**

в **=** **new** ВыражениеРеал**(**сема**);**

далее**();**

**break;**

**case** T**.**СимЛитерал**:**

в **=** **new** ВыражениеСим**(**сема**.**дим\_**);**

далее**();**

**break;**

**case** T**.**Ткст**:**

ткст ткт **=** сема**.**ткт**;**

сим postfix **=** сема**.**pf**;**

далее**();**

**while** **(**сема**.**вид **==** T**.**Ткст**)**

**{**

/+if (postfix == 0)

postfix = сема.pf;

else+/

**if** **(**сема**.**pf **&&** сема**.**pf **!=** postfix**)**

ошибка**(**сема**,** сооб**.**StringPostfixMismatch**);**

ткт**.**length **=** ткт**.**length **-** 1**;** // Exclude '\0'.

ткт **~=** сема**.**ткт**;**

далее**();**

**}**

**switch** **(**postfix**)**

**{**

**case** 'w'**:**

**if** **(**естьНеверныйЮ8**(**ткт**,** начало**))**

**goto** **default;**

в **=** **new** ТекстовоеВыражение**(**drc**.**Unicode**.**вЮ16**(**ткт**));** **break;**

**case** 'd'**:**

**if** **(**естьНеверныйЮ8**(**ткт**,** начало**))**

**goto** **default;**

в **=** **new** ТекстовоеВыражение**(**drc**.**Unicode**.**вЮ32**(**ткт**));** **break;**

**case** 'c'**:**

**default:**

// No checking done в allow for binary данные.

в **=** **new** ТекстовоеВыражение**(**ткт**);** **break;**

**}**

**break;**

**case** T**.**ЛКвСкобка**:**

Выражение**[]** значения**;**

далее**();**

**if** **(!**проверено**(**T**.**ПКвСкобка**))**

**{**

в **=** разборВыраженияПрисвой**();**

**if** **(**проверено**(**T**.**Двоеточие**))**

**goto** LparseAssocArray**;**

**if** **(**проверено**(**T**.**Запятая**))**

значения **=** **[**в**]** **~** разборСпискаВыражений**();**

требуетсяЗакрыв**(**T**.**ПКвСкобка**,** начало**);**

**}**

в **=** **new** ВыражениеЛитералМассива**(**значения**);**

**break;**

LparseAssocArray**:**

Выражение**[]** ключи **=** **[**в**];**

**goto** LenterLoop**;**

**do**

**{**

ключи **~=** разборВыраженияПрисвой**();**

требуется**(**T**.**Двоеточие**);**

LenterLoop**:**

значения **~=** разборВыраженияПрисвой**();**

**}** **while** **(**проверено**(**T**.**Запятая**))**

требуетсяЗакрыв**(**T**.**ПКвСкобка**,** начало**);**

в **=** **new** ВыражениеЛитералАМассива**(**ключи**,** значения**);**

**break;**

**case** T**.**ЛФСкобка**:**

// DelegateLiteral := { Statements }

**auto** телоФунк **=** разборТелаФункции**();**

в **=** **new** ВыражениеЛитералФункции**(**телоФунк**);**

**break;**

**case** T**.**Функция**,** T**.**Делегат**:**

// FunctionLiteral := ("function"|"delegate") Тип? "(" ArgumentList ")" FunctionBody

далее**();** // Skip function or delegate keyword.

Тип типВозврата**;**

Параметры параметры**;**

**if** **(**сема**.**вид **!=** T**.**ЛФСкобка**)**

**{**

**if** **(**сема**.**вид **!=** T**.**ЛСкобка**)** // Optional return тип

типВозврата **=** разборТипа**();**

параметры **=** разборСпискаПараметров**();**

**}**

**auto** телоФунк **=** разборТелаФункции**();**

в **=** **new** ВыражениеЛитералФункции**(**типВозврата**,** параметры**,** телоФунк**);**

**break;**

**case** T**.**Подтвердить**:**

Выражение сооб**;**

требуетсяСледующий**(**T**.**ЛСкобка**);**

в **=** разборВыраженияПрисвой**();**

**if** **(**проверено**(**T**.**Запятая**))**

сооб **=** разборВыраженияПрисвой**();**

требуется**(**T**.**ПСкобка**);**

в **=** **new** ВыражениеПодтверди**(**в**,** сооб**);**

**break;**

**case** T**.**Смесь**:**

требуетсяСледующий**(**T**.**ЛСкобка**);**

в **=** разборВыраженияПрисвой**();**

требуется**(**T**.**ПСкобка**);**

в **=** **new** ВыражениеСмесь**(**в**);**

**break;**

**case** T**.**Импорт**:**

требуетсяСледующий**(**T**.**ЛСкобка**);**

в **=** разборВыраженияПрисвой**();**

требуется**(**T**.**ПСкобка**);**

в **=** **new** ВыражениеИмпорта**(**в**);**

**break;**

**case** T**.**Идтипа**:**

требуетсяСледующий**(**T**.**ЛСкобка**);**

**auto** тип **=** разборТипа**();**

требуется**(**T**.**ПСкобка**);**

в **=** **new** ВыражениеИдТипа**(**тип**);**

**break;**

**case** T**.**Является**:**

далее**();**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

Тип тип**,** типСпец**;**

Идентификатор**\*** идент**;** // optional Идентификатор

Сема**\*** опцСема**,** спецСема**;**

тип **=** разборДекларатора**(**идент**,** да**);**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Двоеточие**,** T**.**Равно**:**

опцСема **=** сема**;**

далее**();**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Типдеф**,**

T**.**Структура**,**

T**.**Союз**,**

T**.**Класс**,**

T**.**Интерфейс**,**

T**.**Перечень**,**

T**.**Функция**,**

T**.**Делегат**,**

T**.**Супер**,**

T**.**Итог**:**

case\_Const\_Invariant**:**

спецСема **=** сема**;**

далее**();**

**break;**

**case** T**.**Конст**,** T**.**Инвариант**:**

**if** **(**возьмиСледщ**()** **!=** T**.**ЛСкобка**)**

**goto** case\_Const\_Invariant**;**

// Fall through. It's a тип.

**default:**

типСпец **=** разборТипа**();**

**}**

**default:**

**}**

ПараметрыШаблона шпарамы**;**

**version(**D2**)**

**{**

// is ( Тип Идентификатор : TypeSpecialization , TemplateParameterList )

// is ( Тип Идентификатор == TypeSpecialization , TemplateParameterList )

**if** **(**идент **&&** типСпец **&&** сема**.**вид **==** T**.**Запятая**)**

шпарамы **=** разборСпискаПараметровШаблона2**();**

**}**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

в **=** **new** ВыражениеЯвляется**(**тип**,** идент**,** опцСема**,** спецСема**,** типСпец**,** шпарамы**);**

**break;**

**case** T**.**ЛСкобка**:**

**if** **(**семаПослеСкобкиЯвляется**(**T**.**ЛФСкобка**))** // Check for "(...) {"

**{** // ( ParameterList ) FunctionBody

**auto** параметры **=** разборСпискаПараметров**();**

**auto** телоФунк **=** разборТелаФункции**();**

в **=** **new** ВыражениеЛитералФункции**(null,** параметры**,** телоФунк**);**

**}**

**else**

**{** // ( Выражение )

**auto** leftParen **=** сема**;**

пропусти**(**T**.**ЛСкобка**);**

в **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

в **=** **new** ВыражениеРодит**(**в**);**

**}**

**break;**

**version(**D2**)**

**{**

**case** T**.**Трэтс**:**

требуетсяСледующий**(**T**.**ЛСкобка**);**

**auto** ид **=** требуетсяИдентификатор**(**сооб**.**ОжидалсяИдентификатор**);**

АргументыШаблона арги**;**

**if** **(**сема**.**вид **==** T**.**Запятая**)**

арги **=** parseTemplateArguments2**();**

**else**

требуется**(**T**.**ПСкобка**);**

в **=** **new** ВыражениеТрактовки**(**ид**,** арги**);**

**break;**

**}**

**default:**

**if** **(**сема**.**интегральныйТип\_ли**)**

**{** // ИнтегральныйТип . Идентификатор

**auto** тип **=** **new** ИнтегральныйТип**(**сема**.**вид**);**

далее**();**

установи**(**тип**,** начало**);**

требуется**(**T**.**Точка**);**

**auto** идент **=** требуетсяИдентификатор**(**сооб**.**ExpectedIdAfterTypeDot**);**

в **=** **new** ВыражениеИдТипаТочка**(**тип**,** идент**);**

**}**

**else** **if** **(**сема**.**спецСема\_ли**)**

**{**

в **=** **new** ВыражениеСпецСема**(**сема**);**

далее**();**

**}**

**else**

**{**

ошибка2**(**ИДС**.**НайденоИноеЧемОжидалось**,** "Выражение"**,** сема**);**

в **=** **new** НелегальноеВыражение**();**

**if** **(!**пробуем**)**

**{** // Insert a dummy сема and don't consume current one.

начало **=** лексер**.**вставьПустуюСемуПеред**(**сема**);**

**this.**предыдущСема **=** начало**;**

**}**

**}**

**}**

установи**(**в**,** начало**);**

**return** в**;**

**}**

Выражение разборВыраженияНов**(**/\*Выражение в\*/**)**

**{**

**auto** начало **=** сема**;**

пропусти**(**T**.**Нов**);**

Выражение**[]** аргиНов**;**

Выражение**[]** аргиКтора**;**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

аргиНов **=** разборАргументов**();**

// ВыражениеНовАнонКласс:

// new (ArgumentList)opt class (ArgumentList)opt SuperClassopt ЦелerfaceClassesopt ClassBody

**if** **(**проверено**(**T**.**Класс**))**

**{**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

аргиКтора **=** разборАргументов**();**

ТипКлассОснова**[]** основы **=** сема**.**вид **!=** T**.**ЛФСкобка **?** разборБазовыхКлассов**(**нет**)** **:** **null** **;**

**auto** деклы **=** разборТелаДефиницииДекларации**();**

**return** установи**(new** ВыражениеНовАнонКласс**(**/\*в, \*/аргиНов**,** основы**,** аргиКтора**,** деклы**),** начало**);**

**}**

// ВыражениеНов:

// NewArguments Тип [ ВыражениеПрисвой ]

// NewArguments Тип ( ArgumentList )

// NewArguments Тип

**auto** тип **=** разборТипа**();**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

аргиКтора **=** разборАргументов**();**

**return** установи**(new** ВыражениеНов**(**/\*в, \*/аргиНов**,** тип**,** аргиКтора**),** начало**);**

**}**

/// Parses a Тип.

Тип разборТипа**()**

**{**

**return** разборБазовогоТипа2**(**разборБазовогоТипа**());**

**}**

Тип разборТипаИдентификатора**()**

**{**

**auto** начало **=** сема**;**

**auto** идент **=** требуетсяИдентификатор**(**сооб**.**ОжидалсяИдентификатор**);**

Тип t**;**

**if** **(**проверено**(**T**.**Не**))** // Идентификатор !( АргументыШаблона )

t **=** **new** ТЭкземплярШаблона**(**идент**,** parseTemplateArguments**());**

**else** // Идентификатор

t **=** **new** ТИдентификатор**(**идент**);**

**return** установи**(**t**,** начало**);**

**}**

Тип разборКвалифицированногоТипа**()**

**{**

**auto** начало **=** сема**;**

Тип тип**;**

**if** **(**сема**.**вид **==** T**.**Точка**)**

тип **=** установи**(new** ТМасштабМодуля**(),** начало**,** начало**);**

**else** **if** **(**сема**.**вид **==** T**.**Типа**)**

тип **=** parseTypeofType**();**

**else**

тип **=** разборТипаИдентификатора**();**

**while** **(**проверено**(**T**.**Точка**))**

тип **=** установи**(new** КвалифицированныйТип**(**тип**,** разборТипаИдентификатора**()),** начало**);**

**return** тип**;**

**}**

Тип разборБазовогоТипа**()**

**{**

**auto** начало **=** сема**;**

Тип t**;**

**if** **(**сема**.**интегральныйТип\_ли**)**

**{**

t **=** **new** ИнтегральныйТип**(**сема**.**вид**);**

далее**();**

**}**

**else**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Идентификатор**,** T**.**Типа**,** T**.**Точка**:**

t **=** разборКвалифицированногоТипа**();**

**return** t**;**

**version(**D2**)**

**{**

**case** T**.**Конст**:**

// const ( Тип )

требуетсяСледующий**(**T**.**ЛСкобка**);**

t **=** разборТипа**();**

требуется**(**T**.**ПСкобка**);**

t **=** **new** ТКонст**(**t**);**

**break;**

**case** T**.**Инвариант**:**

// invariant ( Тип )

требуетсяСледующий**(**T**.**ЛСкобка**);**

t **=** разборТипа**();**

требуется**(**T**.**ПСкобка**);**

t **=** **new** ТИнвариант**(**t**);**

**break;**

**}** // version(D2)

**default:**

ошибка2**(**ИДС**.**НайденоИноеЧемОжидалось**,** "BasicType"**,** сема**);**

t **=** **new** НелегальныйТип**();**

далее**();**

**}**

**return** установи**(**t**,** начало**);**

**}**

Тип разборБазовогоТипа2**(**Тип t**)**

**{**

**while** **(**1**)**

**{**

**auto** начало **=** сема**;**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Умножь**:**

t **=** **new** ТУказатель**(**t**);**

далее**();**

**break;**

**case** T**.**ЛКвСкобка**:**

t **=** разборТипаМассив**(**t**);**

**continue;**

**case** T**.**Функция**,** T**.**Делегат**:**

TOK лекс **=** сема**.**вид**;**

далее**();**

**auto** параметры **=** разборСпискаПараметров**();**

**if** **(**лекс **==** T**.**Функция**)**

t **=** **new** ТФункция**(**t**,** параметры**);**

**else**

t **=** **new** ТДелегат**(**t**,** параметры**);**

**break;**

**default:**

**return** t**;**

**}**

установи**(**t**,** начало**);**

**}**

**assert(**0**);**

**}**

/// Returns да if the сема after the закрывающий parenthesis

/// matches the searched вид.

бул семаПослеСкобкиЯвляется**(**TOK вид**)**

**{**

**assert(**сема**.**вид **==** T**.**ЛСкобка**);**

**auto** следщ **=** сема**;**

**return** пропустиСкобку**(**следщ**)** **==** вид**;**

**}**

/// определено

бул семаПослеСкобкиЯвляется**(**TOK вид**,** ref Сема**\*** следщ**)**

**{**

**assert(**следщ **!is** **null** **&&** следщ**.**вид **==** T**.**ЛСкобка**);**

**return** пропустиСкобку**(**следщ**)** **==** вид**;**

**}**

/// Skips в the сема behind the закрывающий parenthesis.

/// Takes nested parentheses into account.

TOK пропустиСкобку**(**ref Сема**\*** следщ**)**

**{**

**assert(**следщ **!is** **null** **&&** следщ**.**вид **==** T**.**ЛСкобка**);**

// We счёт nested parentheses семы because template types, typeof etc.

// may appear внутри parameter lists. E.g.: (цел x, Foo!(цел) y)

бцел уровень **=** 1**;**

Loop**:**

**while** **(**1**)**

**switch** **(**возьмиПосле**(**следщ**))**

**{**

**case** T**.**ЛСкобка**:**

**++**уровень**;**

**break;**

**case** T**.**ПСкобка**:**

**if** **(--**уровень **==** 0**)**

**return** возьмиПосле**(**следщ**);** // Closing parenthesis found.

**break;**

**case** T**.**КФ**:**

**return** T**.**КФ**;**

**default:**

**}**

**assert(**0**,** "должно быть недоступно"**);**

**}**

/// Parse the массив types after the declarator (C-style.) E.g.: цел a[]

Тип разборСуффиксаДекларатора**(**Тип lhsType**)**

**{**

// The Тип chain should be as follows:

// цел[3]\* Идентификатор [][32]

// <- <- -> -.

// ^-----------------´

// Итогing chain: [][32]\*[3]цел

Тип разборСледующего**()** // Nested function required в accomplish this.

**{**

**if** **(**сема**.**вид **!=** T**.**ЛКвСкобка**)**

**return** lhsType**;** // Всё recursion; return Тип on the левый hand сторона of the Идентификатор.

**auto** начало **=** сема**;**

Тип t**;**

пропусти**(**T**.**ЛКвСкобка**);**

**if** **(**проверено**(**T**.**ПКвСкобка**))**

t **=** **new** ТМассив**(**разборСледующего**());** // [ ]

**else**

**{**

бул успех**;**

Тип разборТипаАМ**()**

**{**

**auto** тип **=** разборТипа**();**

требуется**(**T**.**ПКвСкобка**);**

**return** тип**;**

**}**

**auto** ассоцТип **=** пробуй\_**(&**разборТипаАМ**,** успех**);**

**if** **(**успех**)**

t **=** **new** ТМассив**(**разборСледующего**(),** ассоцТип**);** // [ Тип ]

**else**

**{**

Выражение в **=** разборВыражения**(),** e2**;**

**if** **(**проверено**(**T**.**Срез**))**

e2 **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПКвСкобка**,** начало**);**

t **=** **new** ТМассив**(**разборСледующего**(),** в**,** e2**);** // [ Выражение .. Выражение ]

**}**

**}**

установи**(**t**,** начало**);**

**return** t**;**

**}**

**return** разборСледующего**();**

**}**

Тип разборТипаМассив**(**Тип t**)**

**{**

**auto** начало **=** сема**;**

пропусти**(**T**.**ЛКвСкобка**);**

**if** **(**проверено**(**T**.**ПКвСкобка**))**

t **=** **new** ТМассив**(**t**);**

**else**

**{**

бул успех**;**

Тип разборТипаАМ**()**

**{**

**auto** тип **=** разборТипа**();**

требуется**(**T**.**ПКвСкобка**);**

**return** тип**;**

**}**

**auto** ассоцТип **=** пробуй\_**(&**разборТипаАМ**,** успех**);**

**if** **(**успех**)**

t **=** **new** ТМассив**(**t**,** ассоцТип**);**

**else**

**{**

Выражение в **=** разборВыражения**(),** e2**;**

**if** **(**проверено**(**T**.**Срез**))**

e2 **=** разборВыражения**();**

требуетсяЗакрыв**(**T**.**ПКвСкобка**,** начало**);**

t **=** **new** ТМассив**(**t**,** в**,** e2**);**

**}**

**}**

установи**(**t**,** начало**);**

**return** t**;**

**}**

Тип разборТипаУказательНаФункциюСи**(**Тип тип**,** ref Идентификатор**\*** идент**,** бул опцСписокПарам**)**

**{**

**assert(**тип **!is** **null);**

**auto** начало **=** сема**;**

пропусти**(**T**.**ЛСкобка**);**

тип **=** разборБазовогоТипа2**(**тип**);**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

**{** // Can be nested.

тип **=** разборТипаУказательНаФункциюСи**(**тип**,** идент**,** да**);**

**}**

**else** **if** **(**сема**.**вид **==** T**.**Идентификатор**)**

**{** // The identifier of the function pointer and the declaration.

идент **=** сема**.**идент**;**

далее**();**

тип **=** разборСуффиксаДекларатора**(**тип**);**

**}**

требуетсяЗакрыв**(**T**.**ПСкобка**,** начало**);**

Параметры парамы**;**

**if** **(**опцСписокПарам**)**

парамы **=** сема**.**вид **==** T**.**ЛСкобка **?** разборСпискаПараметров**()** **:** **null;**

**else**

парамы **=** разборСпискаПараметров**();**

тип **=** **new** ТУказательНаФункСи**(**тип**,** парамы**);**

**return** установи**(**тип**,** начало**);**

**}**

Тип разборДекларатора**(**ref Идентификатор**\*** идент**,** бул identOptional **=** нет**)**

**{**

**auto** t **=** разборТипа**();**

**if** **(**сема**.**вид **==** T**.**ЛСкобка**)**

t **=** разборТипаУказательНаФункциюСи**(**t**,** идент**,** да**);**

**else** **if** **(**сема**.**вид **==** T**.**Идентификатор**)**

**{**

идент **=** сема**.**идент**;**

далее**();**

t **=** разборСуффиксаДекларатора**(**t**);**

**}**

**if** **(**идент **is** **null** **&&** **!**identOptional**)**

ошибка2**(**сооб**.**ОжидалсяИдентификаторДекларатора**,** сема**);**

**return** t**;**

**}**

/// Parses a список of AssignExpressions.

/// $(PRE

/// ExpressionList :=

/// ВыражениеПрисвой

/// ВыражениеПрисвой , ExpressionList

/// )

Выражение**[]** разборСпискаВыражений**()**

**{**

Выражение**[]** выражения**;**

**do**

выражения **~=** разборВыраженияПрисвой**();**

**while(**проверено**(**T**.**Запятая**))**

**return** выражения**;**

**}**

/// Parses a список of Аргументы.

/// $(PRE

/// Аргументы :=

/// ( )

/// ( ExpressionList )

/// )

Выражение**[]** разборАргументов**()**

**{**

**auto** leftParen **=** сема**;**

пропусти**(**T**.**ЛСкобка**);**

Выражение**[]** арги**;**

**if** **(**сема**.**вид **!=** T**.**ПСкобка**)**

арги **=** разборСпискаВыражений**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

**return** арги**;**

**}**

/// Parses a ParameterList.

Параметры разборСпискаПараметров**()**

**out(**парамы**)**

**{**

**if** **(**парамы**.**length **>** 1**)**

**foreach** **(**парам**;** парамы**.**элементы**[**0**..**$**-**1**])**

**{**

**if** **(**парам**.**вариадический\_ли**())**

**assert(**0**,** "вариадические аргументы могут появляться только в конце списка параметров."**);**

**}**

**}**

**body**

**{**

**auto** начало **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

**auto** парамы **=** **new** Параметры**();**

**if** **(**проверено**(**T**.**ПСкобка**))**

**return** установи**(**парамы**,** начало**);**

**do**

**{**

**auto** paramBegin **=** сема**;**

КлассХранения кхр**,** stc\_**;**

Тип тип**;**

Идентификатор**\*** идент**;**

Выражение дефЗначение**;**

проц pushParameter**()**

**{**

парамы **~=** установи**(new** Параметр**(**кхр**,** тип**,** идент**,** дефЗначение**),** paramBegin**);**

**}**

**if** **(**проверено**(**T**.**Эллипсис**))**

**{**

кхр **=** КлассХранения**.**Вариадический**;**

pushParameter**();** // тип, идент and дефЗначение will be null.

**break;**

**}**

**while** **(**1**)**

**{** // Parse storage classes.

**switch** **(**сема**.**вид**)**

**{**

**version(**D2**)**

**{**

**case** T**.**Инвариант**:** // D2.0

**if** **(**возьмиСледщ**()** **==** T**.**ЛСкобка**)**

**break;**

stc\_ **=** КлассХранения**.**Инвариант**;**

**goto** Lcommon**;**

**case** T**.**Конст**:** // D2.0

**if** **(**возьмиСледщ**()** **==** T**.**ЛСкобка**)**

**break;**

stc\_ **=** КлассХранения**.**Конст**;**

**goto** Lcommon**;**

**case** T**.**Окончательный**:** // D2.0

stc\_ **=** КлассХранения**.**Окончательный**;**

**goto** Lcommon**;**

**case** T**.**Масштаб**:** // D2.0

stc\_ **=** КлассХранения**.**Масштаб**;**

**goto** Lcommon**;**

**case** T**.**Статический**:** // D2.0

stc\_ **=** КлассХранения**.**Статический**;**

**goto** Lcommon**;**

**}**

**case** T**.**Вхо**:**

stc\_ **=** КлассХранения**.**Вхо**;**

**goto** Lcommon**;**

**case** T**.**Вых**:**

stc\_ **=** КлассХранения**.**Вых**;**

**goto** Lcommon**;**

**case** T**.**Вховых**,** T**.**Реф**:**

stc\_ **=** КлассХранения**.**Реф**;**

**goto** Lcommon**;**

**case** T**.**Отложенный**:**

stc\_ **=** КлассХранения**.**Отложенный**;**

**goto** Lcommon**;**

Lcommon**:**

// Check for redundancy.

**if** **(**кхр **&** stc\_**)**

ошибка2**(**ИДС**.**ПовторяющийсяКлассХранения**,** сема**);**

**else**

кхр **|=** stc\_**;**

далее**();**

**version(**D2**)**

**continue;**

**else**

**break;** // Вхо D1.0 the grammar only allows one storage class.

**default:**

**}**

**break;** // Всё out of inner loop.

**}**

тип **=** разборДекларатора**(**идент**,** да**);**

**if** **(**проверено**(**T**.**Присвоить**))**

дефЗначение **=** разборВыраженияПрисвой**();**

**if** **(**проверено**(**T**.**Эллипсис**))**

**{**

кхр **|=** КлассХранения**.**Вариадический**;**

pushParameter**();**

**break;**

**}**

pushParameter**();**

**}** **while** **(**проверено**(**T**.**Запятая**))**

требуетсяЗакрыв**(**T**.**ПСкобка**,** начало**);**

**return** установи**(**парамы**,** начало**);**

**}**

АргументыШаблона parseTemplateArguments**()**

**{**

АргументыШаблона шарги**;**

**auto** leftParen **=** сема**;**

требуется**(**T**.**ЛСкобка**);**

**if** **(**сема**.**вид **!=** T**.**ПСкобка**)**

шарги **=** разборАргументовШаблона\_**();**

требуетсяЗакрыв**(**T**.**ПСкобка**,** leftParen**);**

**return** шарги**;**

**}**

**version(**D2**)**

**{**

АргументыШаблона parseTemplateArguments2**()**

**{**

пропусти**(**T**.**Запятая**);**

АргументыШаблона шарги**;**

**if** **(**сема**.**вид **!=** T**.**ПСкобка**)**

шарги **=** разборАргументовШаблона\_**();**

**else**

ошибка**(**сема**,** сооб**.**ОжидалсяТипИлиВыражение**);**

требуется**(**T**.**ПСкобка**);**

**return** шарги**;**

**}**

**}** // version(D2)

АргументыШаблона разборАргументовШаблона\_**()**

**{**

**auto** начало **=** сема**;**

**auto** шарги **=** **new** АргументыШаблона**;**

**do**

**{**

Тип parseType\_**()**

**{**

**auto** тип **=** разборТипа**();**

**if** **(**сема**.**вид **==** T**.**Запятая **||** сема**.**вид **==** T**.**ПСкобка**)**

**return** тип**;**

провал\_пробы**();**

**return** **null;**

**}**

бул успех**;**

**auto** typeArgument **=** пробуй\_**(&**parseType\_**,** успех**);**

**if** **(**успех**)**

// TemplateArgument:

// Тип

// Символ

шарги **~=** typeArgument**;**

**else**

// TemplateArgument:

// ВыражениеПрисвой

шарги **~=** разборВыраженияПрисвой**();**

**}** **while** **(**проверено**(**T**.**Запятая**))**

установи**(**шарги**,** начало**);**

**return** шарги**;**

**}**

/// if ( ConstraintExpression )

Выражение разборДополнительногоКонстрейнта**()**

**{**

**if** **(!**проверено**(**T**.**Если**))**

**return** **null;**

требуется**(**T**.**ЛСкобка**);**

**auto** в **=** разборВыражения**();**

требуется**(**T**.**ПСкобка**);**

**return** в**;**

**}**

ПараметрыШаблона разборСпискаПараметровШаблона**()**

**{**

**auto** начало **=** сема**;**

**auto** шпарамы **=** **new** ПараметрыШаблона**;**

требуется**(**T**.**ЛСкобка**);**

**if** **(**сема**.**вид **!=** T**.**ПСкобка**)**

разборСпискаПараметровШаблона\_**(**шпарамы**);**

требуетсяЗакрыв**(**T**.**ПСкобка**,** начало**);**

**return** установи**(**шпарамы**,** начало**);**

**}**

**version(**D2**)**

**{**

ПараметрыШаблона разборСпискаПараметровШаблона2**()**

**{**

пропусти**(**T**.**Запятая**);**

**auto** начало **=** сема**;**

**auto** шпарамы **=** **new** ПараметрыШаблона**;**

**if** **(**сема**.**вид **!=** T**.**ПСкобка**)**

разборСпискаПараметровШаблона\_**(**шпарамы**);**

**else**

ошибка**(**сема**,** сооб**.**ОжидалисьПараметрыШаблона**);**

**return** установи**(**шпарамы**,** начало**);**

**}**

**}** // version(D2)

/// Parses template параметры.

проц разборСпискаПараметровШаблона\_**(**ПараметрыШаблона шпарамы**)**

**{**

**do**

**{**

**auto** paramBegin **=** сема**;**

ПараметрШаблона tp**;**

Идентификатор**\*** идент**;**

Тип типСпец**,** дефТип**;**

проц разборСпецИИлиДефолтнТипа**()**

**{**

// : SpecializationType

**if** **(**проверено**(**T**.**Двоеточие**))**

типСпец **=** разборТипа**();**

// = DefaultType

**if** **(**проверено**(**T**.**Присвоить**))**

дефТип **=** разборТипа**();**

**}**

**switch** **(**сема**.**вид**)**

**{**

**case** T**.**Алиас**:**

// ПараметрАлиасШаблона:

// alias Идентификатор

пропусти**(**T**.**Алиас**);**

идент **=** требуетсяИдентификатор**(**сооб**.**ОжидалсяАлиасПараметраШаблона**);**

разборСпецИИлиДефолтнТипа**();**

tp **=** **new** ПараметрАлиасШаблона**(**идент**,** типСпец**,** дефТип**);**

**break;**

**case** T**.**Идентификатор**:**

идент **=** сема**.**идент**;**

**switch** **(**возьмиСледщ**())**

**{**

**case** T**.**Эллипсис**:**

// ПараметрКортежШаблона:

// Идентификатор ...

пропусти**(**T**.**Идентификатор**);** пропусти**(**T**.**Эллипсис**);**

**if** **(**сема**.**вид **==** T**.**Запятая**)**

ошибка**(**ИДС**.**ПараметрКортежШаблона**);**

tp **=** **new** ПараметрКортежШаблона**(**идент**);**

**break;**

**case** T**.**Запятая**,** T**.**ПСкобка**,** T**.**Двоеточие**,** T**.**Присвоить**:**

// ПараметрТипаШаблона:

// Идентификатор

пропусти**(**T**.**Идентификатор**);**

разборСпецИИлиДефолтнТипа**();**

tp **=** **new** ПараметрТипаШаблона**(**идент**,** типСпец**,** дефТип**);**

**break;**

**default:**

// ПараметрШаблонЗначения:

// Declarator

идент **=** **null;**

**goto** LTemplateValueParameter**;**

**}**

**break;**

**version(**D2**)**

**{**

**case** T**.**Этот**:**

// ПараметрЭтотШаблона

// this ПараметрТипаШаблона

пропусти**(**T**.**Этот**);**

идент **=** требуетсяИдентификатор**(**сооб**.**ОжидалосьИмяДляПараметраШаблонаThis**);**

разборСпецИИлиДефолтнТипа**();**

tp **=** **new** ПараметрЭтотШаблона**(**идент**,** типСпец**,** дефТип**);**

**break;**

**}**

**default:**

LTemplateValueParameter**:**

// ПараметрШаблонЗначения:

// Declarator

Выражение спецЗначение**,** дефЗначение**;**

**auto** типЗначение **=** разборДекларатора**(**идент**);**

// : SpecializationValue

**if** **(**проверено**(**T**.**Двоеточие**))**

спецЗначение **=** разборУсловнВыражения**();**

// = DefaultValue

**if** **(**проверено**(**T**.**Присвоить**))**

дефЗначение **=** разборУсловнВыражения**();**

tp **=** **new** ПараметрШаблонЗначения**(**типЗначение**,** идент**,** спецЗначение**,** дефЗначение**);**

**}**

// Push template parameter.

шпарамы **~=** установи**(**tp**,** paramBegin**);**

**}** **while** **(**проверено**(**T**.**Запятая**))**

**}**

/// Возвращает ткст of a сема printable в the client.

ткст дайПечатный**(**Сема**\*** сема**)**

**{** // TODO: there are some другой семы that have в be handled, в.g. тксты.

**return** сема**.**вид **==** T**.**КФ **?** "КФ" **:** сема**.**исхТекст**;**

**}**

**alias** требуется ожидаемое**;**

/// Requires a сема of вид лекс.

проц требуется**(**TOK лекс**)**

**{**

**if** **(**сема**.**вид **==** лекс**)**

далее**();**

**else**

ошибка2**(**ИДС**.**НайденоИноеЧемОжидалось**,** Сема**.**вТкст**(**лекс**),** сема**);**

**}**

/// Requires the следщ сема в be of вид лекс.

проц требуетсяСледующий**(**TOK лекс**)**

**{**

далее**();**

требуется**(**лекс**);**

**}**

/// Optionally parses an identifier.

/// Возвращает: null or the identifier.

Идентификатор**\*** дополнительныйИдентификатор**()**

**{**

Идентификатор**\*** ид**;**

**if** **(**сема**.**вид **==** T**.**Идентификатор**)**

**(**ид **=** сема**.**идент**),** пропусти**(**T**.**Идентификатор**);**

**return** ид**;**

**}**

Идентификатор**\*** требуетсяИдентификатор**()**

**{**

Идентификатор**\*** ид**;**

**if** **(**сема**.**вид **==** T**.**Идентификатор**)**

**(**ид **=** сема**.**идент**),** пропусти**(**T**.**Идентификатор**);**

**else**

ошибка**(**ИДС**.**НайденоИноеЧемОжидалось**,** "Идентификатор"**,** сема**.**исхТекст**);**

**return** ид**;**

**}**

/// Reports an ошибка if the current сема is not an identifier.

/// Параметры:

/// ошСооб = the ошибка сообщение в be used.

/// Возвращает: null or the identifier.

Идентификатор**\*** требуетсяИдентификатор**(**ткст ошСооб**)**

**{**

Идентификатор**\*** ид**;**

**if** **(**сема**.**вид **==** T**.**Идентификатор**)**

**(**ид **=** сема**.**идент**),** пропусти**(**T**.**Идентификатор**);**

**else**

ошибка**(**сема**,** ошСооб**,** сема**.**исхТекст**);**

**return** ид**;**

**}**

/// Reports an ошибка if the current сема is not an identifier.

/// Параметры:

/// идс = the ошибка сообщение ID в be used.

/// Возвращает: null or the identifier.

Идентификатор**\*** требуетсяИдентификатор**(**ИДС идс**)**

**{**

Идентификатор**\*** ид**;**

**if** **(**сема**.**вид **==** T**.**Идентификатор**)**

**(**ид **=** сема**.**идент**),** пропусти**(**T**.**Идентификатор**);**

**else**

ошибка**(**идс**,** сема**.**исхТекст**);**

**return** ид**;**

**}**

/// Reports an ошибка if the current сема is not an identifier.

/// Возвращает: null or the сема.

Сема**\*** требуетсяИд**()**

**{**

Сема**\*** семаИд**;**

**if** **(**сема**.**вид **==** T**.**Идентификатор**)**

**(**семаИд **=** сема**),** пропусти**(**T**.**Идентификатор**);**

**else**

ошибка**(**ИДС**.**НайденоИноеЧемОжидалось**,** "Идентификатор"**,** сема**.**исхТекст**);**

**return** семаИд**;**

**}**

Сема**\*** требуетсяСемаИд**(**ткст ошСооб**)**

**{**

Сема**\*** семаИд**;**

**if** **(**сема**.**вид **==** T**.**Идентификатор**)**

**(**семаИд **=** сема**),** пропусти**(**T**.**Идентификатор**);**

**else**

**{**

ошибка**(**сема**,** ошСооб**,** сема**.**исхТекст**);**

семаИд **=** лексер**.**вставьПустуюСемуПеред**(**сема**);**

**this.**предыдущСема **=** семаИд**;**

**}**

**return** семаИд**;**

**}**

/// Reports an ошибка if the закрывающий counterpart of a сема is not found.

проц требуетсяЗакрыв**(**TOK закрывающий**,** Сема**\*** открывающий**)**

**{**

**assert(**закрывающий **==** T**.**ПФСкобка **||** закрывающий **==** T**.**ПСкобка **||** закрывающий **==** T**.**ПКвСкобка**);**

**assert(**открывающий **!is** **null);**

**if** **(!**проверено**(**закрывающий**))**

**{**

**auto** место **=** открывающий**.**дайРеальноеПоложение**();**

**auto** открывающийLoc **=** Формат**(**"(открывающий @{},{})"**,** место**.**номСтр**,** место**.**номСтолб**);**

//ошибка(сема, сооб.ExpectedClosing,

//Сема.вТкст(закрывающий), открывающийLoc, дайПечатный(сема));

**}**

**}**

/// Returns да if the ткст ткт has an invalid UTF-8 sequence.

бул естьНеверныйЮ8**(**ткст ткт**,** Сема**\*** начало**)**

**{**

**auto** invalidUTF8Seq **=** Лексер**.**найдиНедействительнуюПоследовательностьУТФ8**(**ткт**);**

**if** **(**invalidUTF8Seq**.**length**)**

ошибка**(**начало**,** сооб**.**НедействительнаяПоследовательностьУТФ8ВТексте**,** invalidUTF8Seq**);**

**return** invalidUTF8Seq**.**length **!=** 0**;**

**}**

/// Forwards ошибка параметры.

проц ошибка**(**Сема**\*** сема**,** ткст форматирСооб**,** **...)**

**{**

error\_**(**сема**,** форматирСооб**,** \_arguments**,** \_argptr**);**

**}**

/// определено

проц ошибка**(**ИДС идс**,** **...)**

**{**

error\_**(this.**сема**,** ДайСооб**(**идс**),** \_arguments**,** \_argptr**);**

**}**

/// определено

проц ошибка2**(**ткст форматирСооб**,** Сема**\*** сема**)**

**{**

ошибка**(**сема**,** форматирСооб**,** дайПечатный**(**сема**));**

**}**

/// определено

проц ошибка2**(**ИДС идс**,** Сема**\*** сема**)**

**{**

ошибка**(**идс**,** дайПечатный**(**сема**));**

**}**

/// определено

проц ошибка2**(**ИДС идс**,** ткст арг**,** Сема**\*** сема**)**

**{**

ошибка**(**идс**,** арг**,** дайПечатный**(**сема**));**

**}**

/// Создаёт отчёт об ошибках и добавляет его в список.

/// Параметры:

/// сема = используется для получения позиции ошибки.

/// форматирСооб = сообщение компилятора об ошибке.

проц error\_**(**Сема**\*** сема**,** ткст форматирСооб**,** TypeInfo**[]** \_arguments**,** base**.**спис\_ва \_argptr**)**

**{**

**if** **(**пробуем**)**

**{**

счётОшибок**++;**

**return;**

**}**

**auto** положение **=** сема**.**дайПоложениеОшибки**();**

**auto** сооб **=** Формат**(**\_arguments**,** \_argptr**,** форматирСооб**);**

**auto** ошибка **=** **new** ОшибкаПарсера**(**положение**,** сооб**);**

ошибки **~=** ошибка**;**

**if** **(**диаг **!is** **null)**

диаг **~=** ошибка**;**

**}**

**}**