CS323 Compiler Project Phase 3

Group: 12110529 CAO Zhezhen, 12110804 FANG Jiawei, 12110817 ZHANG Zhanwei.

Sorted in alphabetical order.

Test Platform

Name	Value
OS	Ubuntu 22.04.2 LTS on Windows 10 x86_64
Bison	bison (GNU Bison) 3.8.2
Flex	flex 2.6.4
libbison-dev	2:3.8.2+dfsg-1build1
gcc	gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Make	GNU Make 4.3. Built for x86_64-pc-linux-gnu
CMake	cmake 3.22.1

Compile and Run

The minimum required Bison version is **3.6**, which allows detailed error information, which allows detailed error information.

In addition, **LLVM Dev Library** is required.

```
mkdir -p build && cd build
cmake ../
make
```

After successful compilation, run

```
bin/splc 0 [file-name]
```

to get a default AST, symbol table and output as `a.ir` under current working directory.

Basic Feature List

Extended Features List

Migration and Re-factorization

```
modules/
├── libspl
│   ├── CMakeLists.txt
│   └── include
│       └── stdargs.h
├── splc
│   ├── CMakeLists.txt
│   ├── include
│   │   ├── AST
│   │   │   ├── AST.hh
│   │   │   ├── ASTBase.hh
│   │   │   ├── ASTCommons.hh
│   │   │   ├── ASTContext.hh
│   │   │   ├── ASTContextManager.hh
│   │   │   ├── ASTProcess.hh
│   │   │   ├── ASTSymbol.hh
│   │   │   ├── DerivedTypes.hh
│   │   │   ├── Expr.hh
│   │   │   ├── SymbolEntry.hh
│   │   │   ├── Type.hh
│   │   │   ├── TypeCheck.hh
│   │   │   ├── TypeContext.hh
│   │   │   └── Value.hh
│   │   ├── Analysis
│   │   │   └── UnusedVariable.hh
│   │   ├── Basic
│   │   │   ├── Specifiers.hh
│   │   │   └── Typetraits.hh
│   │   ├── CodeGen
│   │   │   ├── Builder.hh
│   │   │   ├── OptimizationLevel.hh
│   │   │   └── SimInstGen.hh
│   │   ├── Core
│   │   │   ├── Base.hh
│   │   │   ├── Options.hh
│   │   │   ├── System.hh
│   │   │   ├── Utils
│   │   │   │   ├── CommandLineParser.hh
│   │   │   │   ├── ControlSequence.hh
│   │   │   │   ├── Location.hh
│   │   │   │   ├── LocationWrapper.hh
│   │   │   │   ├── Logging.hh
│   │   │   │   ├── LoggingLevel.hh
│   │   │   │   └── TraceType.hh
│   │   │   ├── Utils.hh
│   │   │   └── splc.hh
│   │   ├── IO
│   │   │   ├── Driver.hh
│   │   │   ├── IOBase.hh
│   │   │   ├── Preprocessor.hh
│   │   │   └── Scanner.hh
│   │   ├── IR
│   │   │   ├── IRBase.hh
│   │   │   ├── IRBuilder.hh
│   │   │   └── IROptimizer.hh
│   │   └── Translation
│   │       ├── TranslationBase.hh
│   │       ├── TranslationContext.hh
│   │       ├── TranslationContextManager.hh
│   │       ├── TranslationManager.hh
│   │       ├── TranslationOption.hh
│   │       ├── TranslationUnit.hh
│   │       └── TranslationUnitProcess.hh
│   └── src
│       ├── AST
│       │   ├── AST.cc
│       │   ├── ASTBase.cc
│       │   ├── ASTBaseProcess.cc
│       │   ├── ASTBaseType.cc
│       │   ├── ASTBaseValue.cc
│       │   ├── ASTContext.cc
│       │   ├── ASTContextManager.cc
│       │   ├── ASTProcess.cc
│       │   ├── ASTSymbol.cc
│       │   ├── CMakeLists.txt
│       │   ├── Expr.cc
│       │   ├── SymbolEntry.cc
│       │   ├── Type.cc
│       │   ├── TypeCheck.cc
│       │   └── Value.cc
│       ├── Analysis
│       │   ├── CMakeLists.txt
│       │   └── UnusedVariable.cc
│       ├── Basic
│       │   ├── CMakeLists.txt
│       │   └── TypeTraits.cc
│       ├── CodeGen
│       │   ├── CMakeLists.txt
│       │   └── SimInstGen.cc
│       ├── Core
│       │   ├── CMakeLists.txt
│       │   ├── Internal.cc
│       │   ├── Options.cc
│       │   ├── System.cc
│       │   ├── Utils
│       │   │   ├── CommandLineParser.cc
│       │   │   └── Logging.cc
│       │   └── Utils.cc
│       ├── IO
│       │   ├── CMakeLists.txt
│       │   ├── Driver.cc
│       │   ├── Lexer.ll
│       │   ├── Parser.yy
│       │   └── Scanner.cc
│       ├── IR
│       │   ├── CMakeLists.txt
│       │   ├── IRBuilder.cc
│       │   ├── IROptimizer.cc
│       │   └── TAC
│       │       ├── TAC.cc
│       │       └── TACBase.cc
│       ├── Translation
│       │   ├── CMakeLists.txt
│       │   ├── TranslationContext.cc
│       │   ├── TranslationContextManager.cc
│       │   ├── TranslationManager.cc
│       │   └── TranslationUnit.cc
│       └── splc.cc
└── ts
    └── CMakeLists.txt

26 directories, 97 files
```

Code Migration

The entire splc implementation has been migrated to C++ platform.

Library Invocation and Encapsulation

The SPL Compiler has been written into pure library form.

All APIs are extracted from the previous C implementation.

- **Native multi-threaded** implementation is allowed, as all classes are abstracted to fully encapsulate the concepts behind them, including:
- **Translation Manager**: Handles translation-related events that will be triggered inside the parser.
    - **Translation Context Manager**: Manages file inclusion/macro expansion and other utilities
- **AST**:
    - **Type System**: Fully encapsulated such that the entire type system does not depend on any external library, nor any other files under the AST module.
        - Primitive types: Signed/Unsigned Integer of `1, 8, 16, 32, 64` bits, pointer type
        - Aggregate Types: Function type, struct type and array type.
        - Each particular semantic type will have only one **instance** under the same **threaded context**, providing both easy comparison through pointers and thread safety.
        - **Generated Types**: Pointer types of various language constructs, available through static factory method and global map (which ensures uniqueness and type identification).
    - **AST Context Manager**: Provides context management, mainly symbol table management.
        - **AST Context**: Provides context support, mainly definition insertion/variable/function lookup.
- **IO**: IO system including parser/lexer that deals directly with file streams are abstracted away from the core components.
- **CodeGen**: Code generation interface.
- **Analysis**: Static analysis interface.
- **IR**: Generating IR and optimizing them.

    - **IROptimizer**: Optimize IRs
    - **IRBuilder**: Build IRs
- **LLVM**: Importing LLVM utilities for support.
- **Core, Core/Utils**: Core utilities that provides logging support and basic facilities, including but not limited to certain compiler hints including `splc_unreachable` to support more aggressive optimization of the splc program.

The code structure indicates that complex operations, such as parsing and AST processing, are encapsulated within separate libraries or modules. The main program is responsible only for orchestrating these components and invoking their respective functionalities.

Complete Type System

We re-implement a complete type system. The instances of the Type class are immutable: once they are created, they are never changed. Also note that only one instance of a particular type is ever created. Thus seeing if two types are equal is a matter of doing a trivial pointer comparison. To enforce that no two equal instances are created, Type instances can only be created via static factory methods in class Type and in derived classes.

Optimization

Including constant propagation and peephole optimization.

Appendix: Full Grammar

Grammar has been refactored to suit the IR generation. Namely:

- **CallExpr, SubscriptExpr, DerefExpr, AddrOfExpr**: to ease the generation of IR instructions, and provide much fine-grained facilities to improve enduser experience.

```
%skeleton "lalr1.cc"
%require  "3.8.2"
%define parse.trace // This is required for runtime traces. For example, symbol_name.
%define parse.assert

%code requires{
    // Code section there will be placed directly inside `IO/Parser.hh`.
    #include "Core/Utils/LocationWrapper.hh"
    #include "Core/Base.hh"
    namespace splc {

    class AST;
    using PtrAST = Ptr<AST>;
    class TranslationManager;

    namespace IO {
        class Driver;
        class Scanner;
        class Parser;
    } // namespace splc::IO

    } // namespace splc

    // TODO: finish all error recovery
}

%parse-param
%parse-param
%parse-param

%code{
    #include <iostream>
    #include <cstdlib>
    #include <fstream>

    // include for all driver functions
    #include "Core/splc.hh"

    #include "IO/Driver.hh"

    #include "AST/AST.hh"
    #include "Translation/TranslationManager.hh"

    using SymbolType = splc::ASTSymbolType;

    #undef yylex
    #define yylex scanner.yylex

    // TODO: allow context pushing/switch
    // TODO: error recovery (instead of applying the default one in Bison)
}

//===----------------------------------------------------------------------===//
//                               API Settings
//===----------------------------------------------------------------------===//
%define api.namespace
%define api.parser.class
// %define api.header.include
// %define api.location.file "../../include/Core/Utils/location.hh"
%define api.location.type

%define api.symbol.prefix  // The empty prefix is generally invalid, but there is
namespace in C++.
%define api.value.type
%locations



//===----------------------------------------------------------------------===//
//                              Token Definitions
//===----------------------------------------------------------------------===//

//===----------------------------------------------------------------------===//
//===-Storage Qualifiers
%token KwdAuto KwdExtern KwdRegister KwdStatic KwdTypedef
```

```
//===---------------------------------------------------------------------===//
//===-Type Qualifiers
%token KwdConst KwdRestrict KwdVolatile

//===---------------------------------------------------------------------===//
//===-Function Specifiers
%token KwdInline

//===---------------------------------------------------------------------===//
//===-Primitive Type Specifiers
%token VoidTy IntTy SignedTy UnsignedTy LongTy FloatTy DoubleTy CharTy
%token KwdEnum

//===---------------------------------------------------------------------===//
//===-Aggregate Type Specifier
%token KwdStruct KwdUnion

//===---------------------------------------------------------------------===//
//===-Keywords
// Flow Controls
%token KwdIf KwdElse KwdSwitch
%token KwdWhile KwdFor KwdDo
// Labels
%token KwdDefault KwdCase
// Jumps
%token KwdGoto KwdContinue KwdBreak KwdReturn

//===---------------------------------------------------------------------===//
//===-IDs
%token ID TypedefID

//===---------------------------------------------------------------------===//
//===-Operators
// Assignments
%token OpAssign
%token OpMulAssign OpDivAssign OpModAssign OpPlusAssign OpMinusAssign
%token OpLShiftAssign OpRShiftAssign OpBAndAssign OpBXorAssign OpBOrAssign

// Conditional
%token OpAnd OpOr OpNot
%token OpLT OpLE OpGT OpGE OpNE OpEQ
%token OpQMark OpColon

// Arithmetics
%token OpLShift OpRShift
%token OpBAnd OpBOr OpBNot OpBXor

%token OpDPlus OpDMinus OpPlus OpMinus OpAstrk OpDiv OpMod

// Builtin
%token OpDot OpRArrow
%token OpSizeOf
%token OpLSB OpRSB
```

```
// Misc
%token OpComma OpEllipsis

//===----------------------------------------------------------------------===//
//===-Punctuators
%token PSemi
%token PLC PRC
%token PLP PRP

//===-Literals
%token UIntLiteral SIntLiteral FloatLiteral CharLiteral StrUnit

//===----------------------------------------------------------------------===//
//                           Additional Tokens
//===----------------------------------------------------------------------===//
%token SubscriptExpr CallExpr AccessExpr
%token ExplicitCastExpr
%token AddrOfExpr DerefExpr
%token SizeOfExpr

//===----------------------------------------------------------------------===//
//                         Precedence Specification
//===----------------------------------------------------------------------===//
%precedence KwdThen
%precedence KwdElse

%precedence DecltrPrec
%precedence FuncDeclPrec

%left OpComma
%right OpAssign OpMulAssign OpDivAssign OpModAssign OpPlusAssign OpMinusAssign
OpLShiftAssign OpRShiftAssign OpBAndAssign OpBXorAssign OpBOrAssign
%right OpQMark OpColon
%left OpOr
%left OpAnd
%left OpBOr
%left OpBXor
%left OpBAnd
%left OpLT OpLE OpGT OpGE OpNE OpEQ
%left OpPlus OpMinus
%left OpAstrk OpDiv OpMod
%right OpUnaryPrec
%right OpNot OpBNot OpDPlus OpDMinus OpSizeOf
%left PLParen PRParen PLSBracket PRSBracket OpDot

//===----------------------------------------------------------------------===//
//                              Test Specification
//===----------------------------------------------------------------------===//

//===----------------------------------------------------------------------===//
//                           Production Definitions
//===----------------------------------------------------------------------===//
%%
/* Entire translation unit */
ParseRoot:
    TransUnit {
        transMgr.setRootNode($TransUnit);
        SPLC_LOG_DEBUG(&@TransUnit, true) << "completed parsing";

        $TransUnit->setASTContext(transMgr.getCurrentASTContext());
        transMgr.popASTContext();
    }
    ;

TransUnit:
      ExternDeclList
    |
    ;

/* External definition list: Recursive definition */
ExternDeclList:
      ExternDecl
    | ExternDeclList ExternDecl
    ;

/* External definition list: A single unit of one of . */
ExternDecl:
      PSemi
    | Decl
    | FuncDef
    | FuncDecl
    ;

DeclSpec:
      StorageSpec
    | TypeSpec
    | TypeQual
    | FuncSpec
    | DeclSpec TypeSpec
    | DeclSpec StorageSpec
    | DeclSpec TypeQual
    | DeclSpec FuncSpec
    ;

StorageSpec:
      KwdAuto
    | KwdExtern
    | KwdRegister
    | KwdStatic
    | KwdTypedef
    ;

SpecQualList:
      TypeSpec
    | TypeQual
    | SpecQualList TypeSpec
    | SpecQualList TypeQual
    ;
```

```
TypeSpec:
      BuiltinTypeSpec
    /* | identifier  */
    | StructOrUnionSpec
    | EnumSpec
    | TypedefID
    ;

FuncSpec:
      KwdInline
    ;

TypeQual:
      KwdConst
    | KwdRestrict
    | KwdVolatile
    ;

TypeName:
      SpecQualList
    | SpecQualList AbsDecltr
    ;

BuiltinTypeSpec:
      VoidTy
    | IntTy
    | FloatTy
    | DoubleTy
    | CharTy
    | SignedTy
    | UnsignedTy
    | LongTy
    ;

AbsDecltr:
      PtrDecltr
    | PtrDecltr DirAbsDecltr {
        // Let PtrDecltr become the parent of this node.
        auto ptrDeclRoot = ASTHelper::getPtrDeclEndPoint(*$1);
        ptrDeclRoot->addChild($2);
        $$ = transMgr.makeAST<AST>(SymbolType::AbsDecltr, @$, ptrDeclRoot);
    }
    ;

DirAbsDecltr:
      PLP AbsDecltr PRP
    | DirAbsDecltr OpLSB AssignExpr OpRSB
    | DirAbsDecltr OpLSB OpRSB
    | DirAbsDecltr PLP ParamList PRP
    | PLP ParamList PRP

    | DirAbsDecltr OpLSB error
    | DirAbsDecltr OpRSB
    ;
```

```
/* Specify a structure */
StructOrUnionSpec:
      StructOrUnion IDWrapper
    | StructOrUnion StructDeclBody
    | StructOrUnion IDWrapper StructDeclBody
    ;

StructOrUnion:
      KwdStruct
    | KwdUnion
    ;

StructDeclBody:
      PLC PRC
    | PLC StructDeclList PRC

    | PLC error
    | PLC StructDeclList error
    ;

StructDeclList:
      StructDecl
    | StructDeclList StructDecl
    ;

StructDecl:
      SpecQualList PSemi
    | SpecQualList StructDecltrList PSemi

    | SpecQualList error
    | SpecQualList StructDecltrList error
    ;

StructDecltrList:
      StructDecltr
    | StructDecltrList OpComma StructDecltr

    | StructDecltrList OpComma error
    ;

StructDecltr:
      Decltr
    | OpColon ConstExpr
    | Decltr OpColon ConstExpr

    | OpColon error
    | Decltr OpColon error
    ;

EnumSpec:
      KwdEnum IDWrapper
    | KwdEnum EnumBody
    | KwdEnum IDWrapper EnumBody

    | KwdEnum error
    ;

EnumBody:
      PLC PRC
    | PLC EnumeratorList PRC
    | PLC EnumeratorList OpComma PRC

    | PLC error
    | PLC EnumeratorList error
    ;

EnumeratorList:
      Enumerator
    | EnumeratorList OpComma Enumerator

    | OpComma Enumerator
    ;

Enumerator:
      EnumConst
    | EnumConst OpAssign ConstExpr

    | EnumConst OpAssign error
    ;

EnumConst:
      IDWrapper
    ;

/* Single variable declaration */
Decltr:
      DirDecltr
    | PtrDecltr DirDecltr  {
        // Let PtrDecltr become the parent of this node.
        auto ptrDeclEndPoint = ASTHelper::getPtrDeclEndPoint(*$1);
        ptrDeclEndPoint->addChild($2);
        $$ = transMgr.makeAST<AST>(SymbolType::Decltr, @$, $1);
    }
    ;

DirDecltr:
      IDWrapper
    | WrappedDirDecltr
    | DirDecltr OpLSB AssignExpr OpRSB
    | DirDecltr OpLSB OpRSB
    | WrappedDirDecltr PLP ParamList PRP {
          $$ = transMgr.makeAST<AST>(SymbolType::DirDecltr, @$, $1, $3);
      }
    | DirDecltr OpLSB AssignExpr error
    /* | direct-declarator error   */
    | DirDecltr OpRSB
    ;

WrappedDirDecltr:
      PLP Decltr PRP
    ;

PtrDecltr:
      OpAstrk
    | OpAstrk TypeQualList
    | OpAstrk PtrDecltr
    | OpAstrk TypeQualList PtrDecltr
    ;

TypeQualList:
      TypeQual
    | TypeQualList TypeQual
    ;

/* Definition: List of definitions. Recursive definition. */
/* declaration-list:
      declaration
    | declaration-list declaration
    ; */

/* Definition: Base */
Decl:
      DirDecl PSemi

    | DirDecl error
    ;

DirDecl:
      DeclSpec
    | DeclSpec InitDecltrList
    ;

/* Definition: Declaration of multiple variable.  */
InitDecltrList:
      InitDecltr
    | InitDecltrList OpComma InitDecltr

    | InitDecltrList OpComma
    | OpComma InitDecltr
    | OpComma
    ;

/* Definition: Single declaration unit. */
InitDecltr:
      Decltr
    | Decltr OpAssign Initializer

    | Decltr OpAssign error
    ;

Initializer:
      AssignExpr
    | PLC InitializerList PRC
    | PLC InitializerList OpComma PRC
    | PLC InitializerList error
    ;

InitializerList:
      Initializer
    | Designation Initializer
    | InitializerList OpComma Designation Initializer
    | InitializerList OpComma Initializer

    | Designation error
    | InitializerList OpComma error
    ;

Designation:
      DesignatorList OpAssign
    ;

DesignatorList:
      Designator
    | DesignatorList Designator
    ;

Designator:
      OpLSB ConstExpr OpRSB
    | OpDot IDWrapper

    | OpLSB ConstExpr error
    | OpDot error
    ;

FuncDef:
      DeclSpec FuncDecltr CompStmt
    | FuncDecltr CompStmt {
          SPLC_LOG_WARN(&@1, true) << "function is missing a specifier and will default
to 'int'";
          auto declSpec = ASTHelper::makeDeclSpecifierTree(Location, SymbolType::IntTy);
          $$ = transMgr.makeAST<AST>(SymbolType::FuncDef, @$, declSpec, $1, $2);
          transMgr.tryRegisterSymbol($$);
      }
    | DeclSpec FuncDecltr error
    ;

FuncDecl:
      FuncDecltr PSemi {
          SPLC_LOG_WARN(&@1, true) << "function is missing a specifier and will default
to 'int'";
          auto declSpec = ASTHelper::makeDeclSpecifierTree(Location, SymbolType::IntTy);
          $$ = transMgr.makeAST<AST>(SymbolType::FuncDecl, @$, declSpec, $1);
           transMgr.tryRegisterSymbol($$);
      }
    | DeclSpec FuncDecltr PSemi
    ;


/* Function: Function name and body. */
```

```
FuncDecltr:
      DirFuncDecltr
    | PtrDecltr DirFuncDecltr {
        // Let PtrDecltr become the parent of this node.
        auto ptrDeclRoot = ASTHelper::getPtrDeclEndPoint(*$1);
        ptrDeclRoot->addChild($2);
        $$ = transMgr.makeAST<AST>(SymbolType::FuncDecltr, @$, ptrDeclRoot);
    }
    ;

DirFuncDecltr:
      DirDecltrForFunc PLP ParamTypeList PRP
    /* | direct-declarator-for-function PLP PRP  */

    /* | direct-declarator-for-function PLP error  */
    | DirDecltrForFunc PLP ParamTypeList error
    /* | direct-declarator-for-function PLP error  */

    | PLP ParamTypeList PRP
    /* | PLP PRP  */
    ;

DirDecltrForFunc:
      IDWrapper
    ;

/* List of variables names */
ParamTypeList:
      ParamList
    | ParamList OpComma OpEllipsis
    ;

ParamList:

    | ParamDecltr
    | ParamList OpComma ParamDecltr

    | ParamList OpComma error
    ;

/* Parameter declaration */
ParamDecltr:
      DeclSpec Decltr
    | DeclSpec AbsDecltr
    | DeclSpec

    /* | error  */
    ;

/* Compound statement: A new scope. */
CompStmt:
      /* PLC general-statement-list PRC */
      PLC GeneralStmtList PRC
    | PLC PRC
    | PLC GeneralStmtList error
    | PLC error
    ;

/* wrapper for C99 standard for statements */
GeneralStmtList:
      Stmt
    | Decl
    /* | FuncDecl  */
    | GeneralStmtList Stmt
    | GeneralStmtList Decl
    ;

/* Statement: List of statements. Recursive definition. */
/* statement-list:
      statement
    | statement-list statement
    ; */

/* Statement: A single statement. */
Stmt: // TODO: use hierarchy
      PSemi
    | CompStmt
    | ExprStmt
    | SelStmt
    | IterStmt
    | LabeledStmt
    | JumpStmt

    /* | error PSemi  */
    ;

ExprStmt:
      Expr PSemi
    | Expr error
    ;

SelStmt:
      KwdIf PLP Expr PRP Stmt %prec KwdThen

    | KwdIf error PRP Stmt %prec KwdThen
    | KwdIf PLP PRP Stmt %prec KwdThen
    | KwdIf PLP Expr PRP error %prec KwdThen
    | KwdIf PLP PRP error %prec KwdThen

    | KwdIf PLP Expr PRP Stmt KwdElse Stmt %prec KwdElse

    | KwdIf error PRP Stmt KwdElse Stmt %prec KwdElse
    | KwdIf PLP Expr PRP Stmt KwdElse error %prec KwdElse
    | KwdIf PLP PRP Stmt KwdElse Stmt %prec KwdElse
    | KwdIf PLP PRP Stmt KwdElse error %prec KwdElse
    | KwdIf PLP Expr error %prec KwdElse
    | KwdElse Stmt

    | KwdSwitch PLP Expr PRP Stmt
    /* | KwdSwitch PLP expression statement  */
    | KwdSwitch error PRP Stmt
    ;

LabeledStmt:
      IDWrapper OpColon Stmt
    | KwdCase ConstExpr OpColon Stmt
    | KwdDefault OpColon Stmt

    | OpColon Stmt
    ;

JumpStmt:
      KwdGoto IDWrapper PSemi
    | KwdContinue PSemi
    | KwdBreak PSemi
    | KwdReturn Expr PSemi
    | KwdReturn PSemi

    | KwdReturn Expr error
    | KwdReturn error
    ;

IterStmt:
      KwdWhile PLP Expr PRP Stmt
    | KwdWhile error PRP Stmt
    | KwdWhile PLP Expr PRP error
    | KwdWhile PLP Expr error

    | KwdDo Stmt KwdWhile PLP Expr PRP PSemi
    | KwdDo Stmt KwdWhile PLP error PSemi

    | KwdFor PLP ForLoopBody PRP Stmt
    | KwdFor PLP ForLoopBody PRP error
    | KwdFor PLP ForLoopBody error
    ;

ForLoopBody: // TODO: add constant expressions
      InitExpr PSemi Expr PSemi Expr

    | PSemi Expr PSemi Expr
    | InitExpr PSemi Expr PSemi
    | InitExpr PSemi PSemi Expr

    | PSemi Expr PSemi
    | PSemi PSemi Expr
    /* | definition PSemi  */
    | InitExpr PSemi PSemi

    | PSemi PSemi
    ;

ConstExpr:
      CondExpr
    ;
```

```
Constant:
      UIntLiteral
    | SIntLiteral
    | FloatLiteral
    | CharLiteral
    /* | StrUnit  */
    ;

PrimaryExpr:
      IDWrapper
    | Constant
    | StringLiteral
    | PLP Expr PRP

    | PLP Expr error
    /* | PLP expression  */
    ;

PostfixExpr:
      PrimaryExpr
    | PostfixExpr OpLSB Expr OpRSB
    | PostfixExpr PLP ArgList PRP
    /* | postfix-expression PLP PRP  */
    | PostfixExpr MemberAcessOp IDWrapper
    | PostfixExpr OpDPlus
    | PostfixExpr OpDMinus
    | PLP TypeName PRP PLC InitializerList PRC
    | PLP TypeName PRP PLC InitializerList OpComma PRC

    | PostfixExpr OpLSB Expr error
    | PostfixExpr PLP ArgList error
    | PostfixExpr MemberAcessOp
    | OpRArrow IDWrapper
    | PLP TypeName PRP PLC InitializerList error
    ;

MemberAcessOp:
      OpDot
    | OpRArrow
    ;

UnaryExpr:
      PostfixExpr
    | OpDPlus UnaryExpr
    | OpDMinus UnaryExpr
    | OpBAnd CastExpr %prec OpUnaryPrec
    | OpAstrk CastExpr %prec OpUnaryPrec
    | UnaryArithOp CastExpr %prec OpUnaryPrec
    | OpSizeOf UnaryExpr
    | OpSizeOf PLP TypeName PRP

    | OpBAnd error
    | OpAstrk error
    | OpBNot error
    | OpNot error
    | OpDPlus error
    | OpDMinus error
    | OpSizeOf error
    /* | OpSizeOf PLP unary-expression PRP  */
    ;

UnaryArithOp: /* Take the default behavior, that is, `$$ = $1` */
      OpPlus
    | OpMinus
    | OpBNot
    | OpNot
    ;


CastExpr:
      UnaryExpr
    | PLP TypeName PRP CastExpr

    | PLP TypeName PRP error
    | PLP TypeName error
    ;

MulExpr:
      CastExpr
    | MulExpr MulOp CastExpr

    | MulExpr MulOp error
    | DivOp CastExpr
    ;

MulOp:
      OpAstrk
    | DivOp
    ;

DivOp:
      OpDiv
    | OpMod
    ;

AddExpr:
      MulExpr
    | AddExpr AddOp MulExpr

    | AddExpr AddOp error
    ;

AddOp:
      OpPlus
    | OpMinus
    ;

ShiftExpr:
      AddExpr
    | ShiftExpr ShiftOp AddExpr

    | ShiftExpr ShiftOp error
    | ShiftOp AddExpr
    ;

ShiftOp:
      OpLShift
    | OpRShift
    ;

RelExpr:
      ShiftExpr
    | RelExpr RelOp ShiftExpr

    | RelExpr RelOp error
    | RelOp ShiftExpr
    ;

RelOp:
      OpLT
    | OpGT
    | OpLE
    | OpGE
    ;

EqualityExpr:
      RelExpr
    | EqualityExpr EqualityOp RelExpr

    | EqualityExpr EqualityOp error
    | EqualityOp RelExpr
    ;

EqualityOp:
      OpEQ
    | OpNE
    ;

OpBAndExpr:
      EqualityExpr
    | OpBAndExpr OpBAnd EqualityExpr

    | OpBAndExpr OpBAnd error
    ;

OpBXorExpr:
      OpBAndExpr
    | OpBXorExpr OpBXor OpBAndExpr

    | OpBXorExpr OpBXor error
    | OpBXor OpBAndExpr
    ;

OpBOrExpr:
      OpBXorExpr
    | OpBOrExpr OpBOr OpBXorExpr

    | OpBOrExpr OpBOr error
    | OpBOr OpBXorExpr
    ;

LogicalOpAndExpr:
      OpBOrExpr
    | LogicalOpAndExpr OpAnd OpBOrExpr

    | LogicalOpAndExpr OpAnd error
    | OpAnd OpBOrExpr
    ;

LogicalOpOrExpr:
      LogicalOpAndExpr
    | LogicalOpOrExpr OpOr LogicalOpAndExpr

    | LogicalOpOrExpr OpOr error
    | OpOr LogicalOpAndExpr
    ;

CondExpr:
      LogicalOpOrExpr
    | LogicalOpOrExpr OpQMark Expr OpColon CondExpr

    | LogicalOpOrExpr OpQMark OpColon CondExpr
    | LogicalOpOrExpr OpQMark Expr OpColon
    | OpQMark error
    ;

AssignExpr:
      CondExpr

    | CondExpr AssignOp AssignExpr
    | CondExpr AssignOp error
    | AssignOp AssignExpr

    /* | unary-expression assignment-operator assignment-expression  */
    /* | unary-expression assignment-operator error  */
    /* | assignment-operator assignment-expression  */
    ;

AssignOp: /* Use the default behavior to pass the value */
      OpAssign
    | OpMulAssign
    | OpDivAssign
    | OpModAssign
    | OpPlusAssign
    | OpMinusAssign
    | OpLShiftAssign
    | OpRShiftAssign
    | OpBAndAssign
    | OpBXorAssign
    | OpBOrAssign
    ;

/* expressions */
Expr:
      AssignExpr
    | Expr OpComma AssignExpr

    | Expr OpComma error
    | OpComma AssignExpr
    ;

InitExpr:
      Expr
    | DirDecl
    ;

/* Argument: List of arguments */
ArgList:

    | ArgList OpComma AssignExpr
    | AssignExpr

    | ArgList OpComma error
    /* | error  */
    ;

/* String intermediate expression. Allowing concatenation of strings. */
StringLiteral:
      StrUnit
    | StringLiteral StrUnit
    ;

IDWrapper:
      ID
    ;
%%


void splc::IO::Parser::error(const location_type &l, const std::string &err_message)
{
    SPLC_LOG_ERROR(&l, true) << err_message;
}
```