并行编译与优化

Advanced Compiler Technology

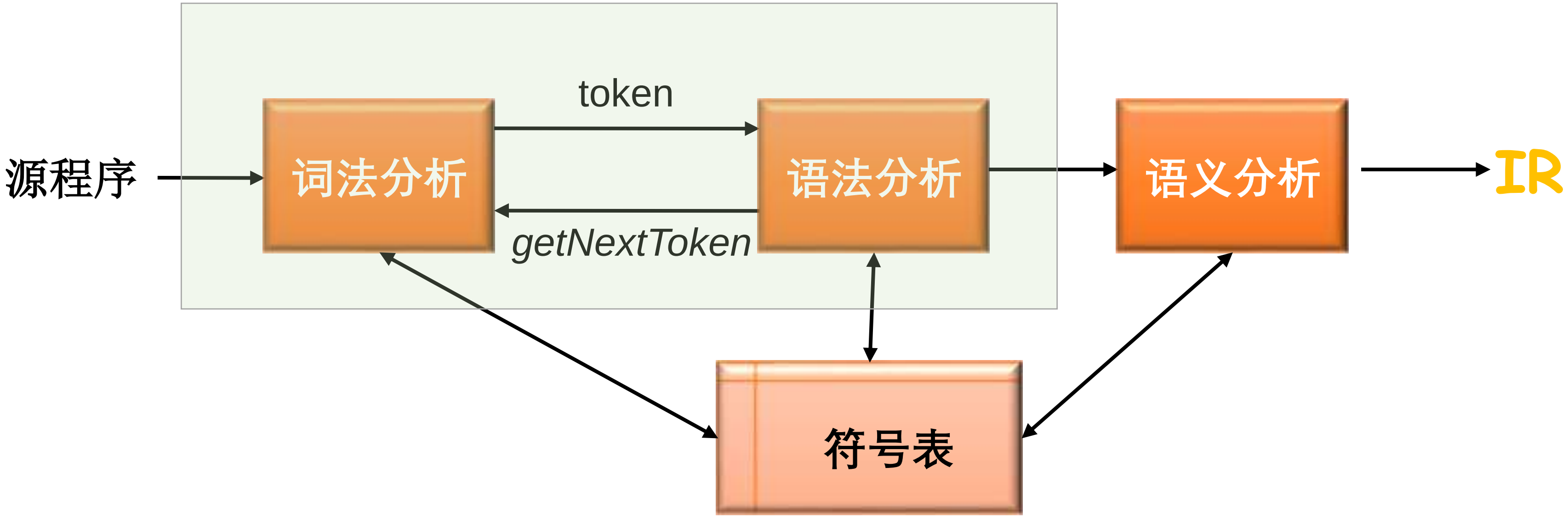
计算机研究所编译室

Experiment One: Implement SysY Lexer/Parser with ANTLR

实验1：用ANTLR实现SysY词法/语法分析器

复习：编译器前端

- 前端
  - 扫描程序，识别合法程序
  - 给出恰当的警告/错误信息
  - 生成中间表示代码（IR）



实验内容

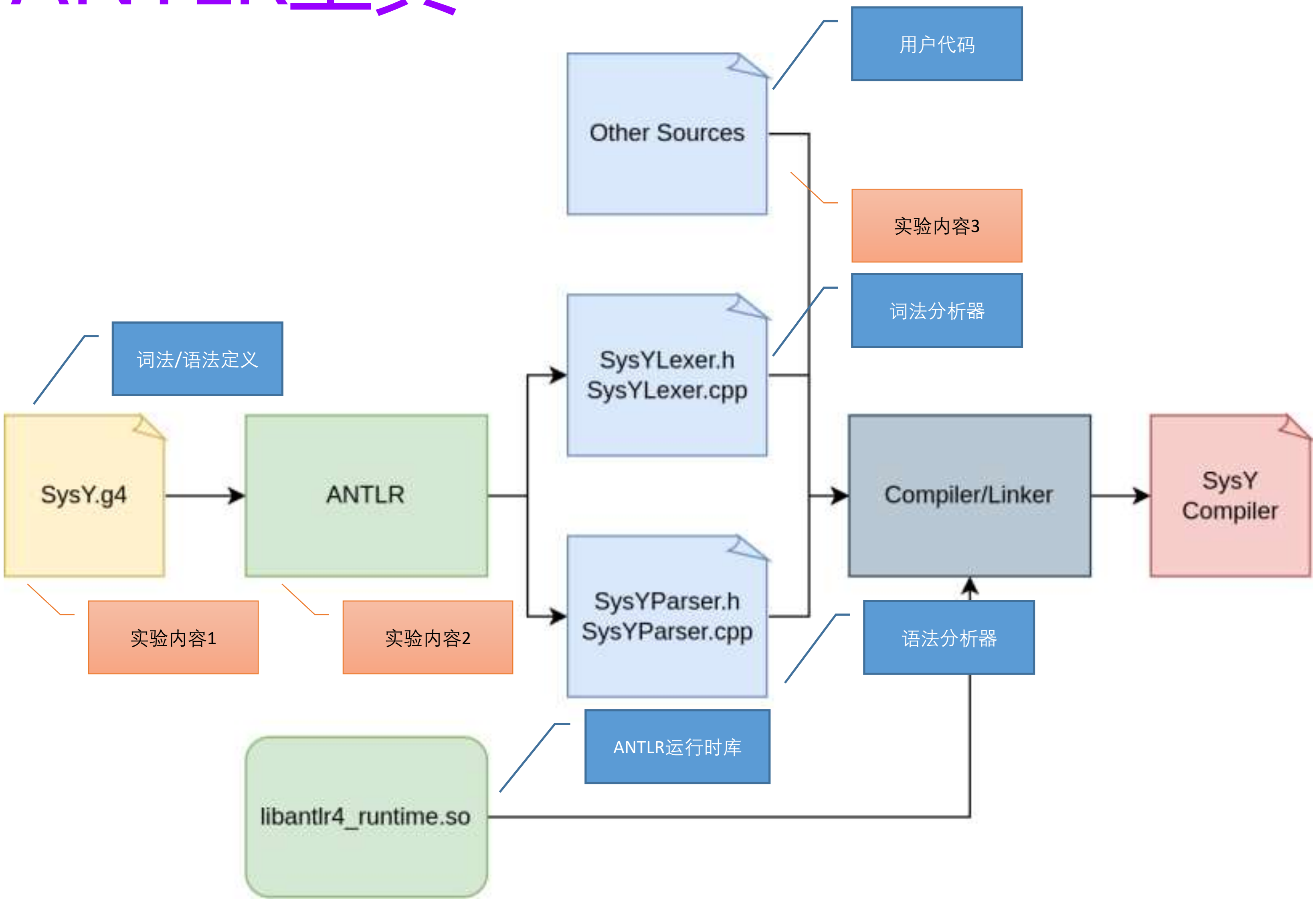
- 定义SysY语言的词法/语法规范
- 使用ANTLR工具生成SysY语言的词法/语法分析器
- 实现SysY语言格式化器（进阶内容）

ANTLR工具

- ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. **From a grammar, ANTLR generates a parser that can build and walk parse trees**.
- 支持Java、C++、Go、C#、Python等多种目标编程语言

一种比Lex/Yacc更加现代的前端工具

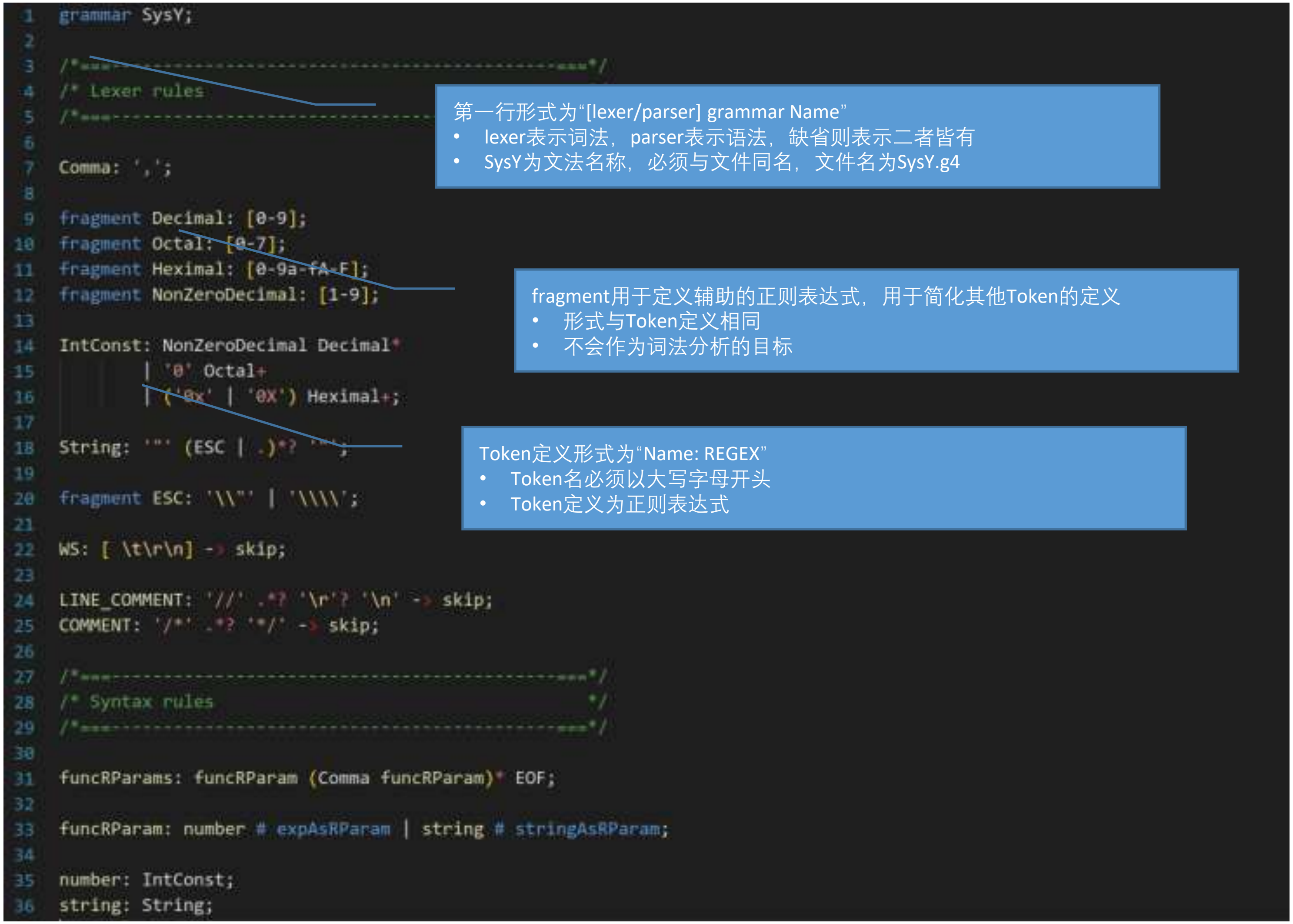
ANTLR工具


用户代码
Other Sources
实验内容3
词法分析器
词法/语法定义
SysYLexer.h
SysYLexer.cpp
SysY.g4
ANTLR
Compiler/Linker
SysY
Compiler
SysYParser.h
SysYParser.cpp
实验内容1
实验内容2
语法分析器
ANTLR运行时库
libantlr4_runtime.so

实验内容1

定义SysY词法/语法规范

SysY词法定义



```
grammar SysY;

/*===-----------------------------------------------===*/
/* Lexer rules
/*===------------------------------

Comma: ',';

fragment Decimal: [0-9];
fragment Octal: [0-7];
fragment Heximal: [0-9a-fA-F];
fragment NonZeroDecimal: [1-9];

IntConst: NonZeroDecimal Decimal*
        | '0' Octal+
        | ('0x' | '0X') Heximal+;

String: '"' (ESC | .)*? '"';

fragment ESC: '\\"' | '\\\\';

WS: [ \t\r\n] -> skip;

LINE_COMMENT: '//' .*? '\r'? '\n' -> skip;
COMMENT: '/*' .*? '*/' -> skip;

/*===-----------------------------------------------===*/
/* Syntax rules                                        */
/*===-----------------------------------------------===*/

funcRParams: funcRParam (Comma funcRParam)* EOF;

funcRParam: number # expAsRParam | string # stringAsRParam;

number: IntConst;
string: String;
```

第一行形式为“[lexer/parser] grammar Name”

- lexer表示词法，parser表示语法，缺省则表示二者皆有
- SysY为文法名称，必须与文件同名，文件名为SysY.g4

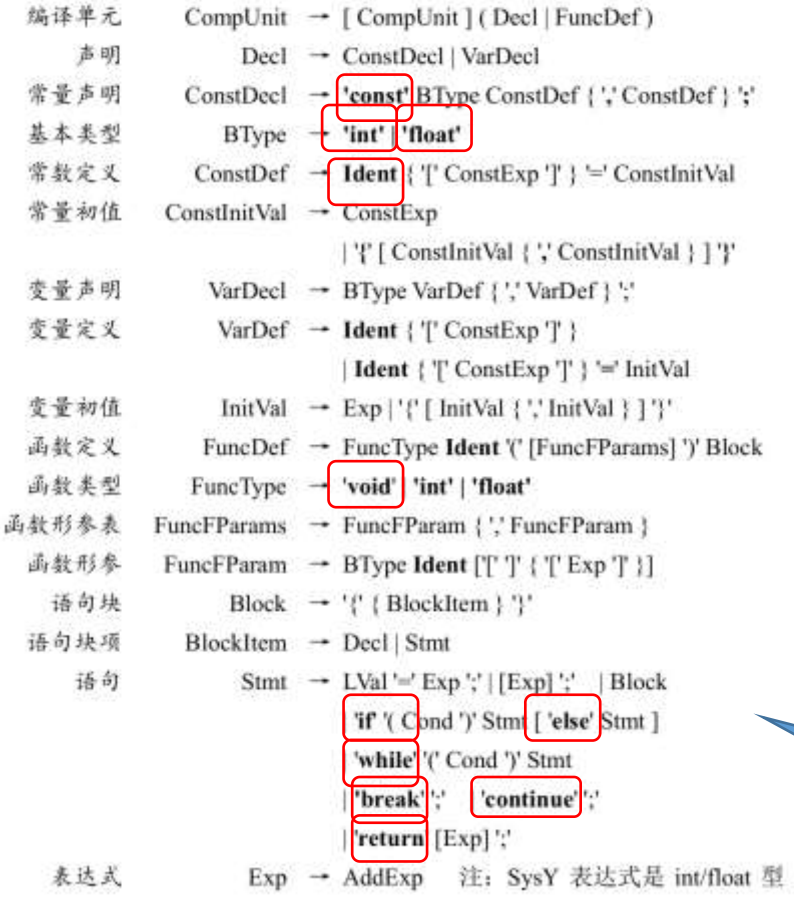
fragment用于定义辅助的正则表达式，用于简化其他Token的定义

- 形式与Token定义相同
- 不会作为词法分析的目标

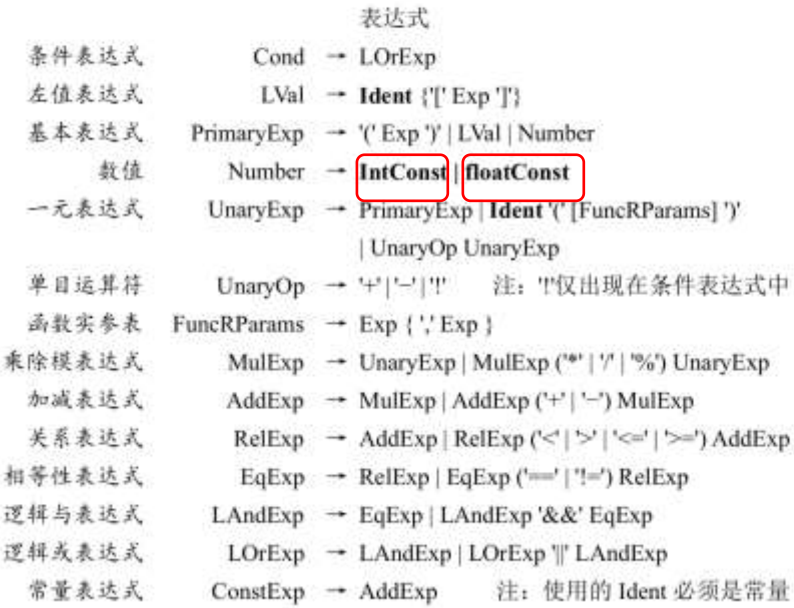
Token定义形式为“Name: REGEX”

- Token名必须以大写字母开头
- Token定义为正则表达式

SysY语法规范



编译单元	CompUnit	→	[CompUnit] (Decl | FuncDef)
声明	Decl	→	ConstDecl | VarDecl
常量声明	ConstDecl	→	**'const'** BType ConstDef { ',' ConstDef } ';'
基本类型	BType	→	**'int'** | **'float'**
常数定义	ConstDef	→	**Ident** { '[' ConstExp ']' } '=' ConstInitVal
常量初值	ConstInitVal	→	ConstExp
			| '{' [ConstInitVal { ',' ConstInitVal }] '}'
变量声明	VarDecl	→	BType VarDef { ',' VarDef } ';'
变量定义	VarDef	→	**Ident** { '[' ConstExp ']' }
			| **Ident** { '[' ConstExp ']' } '=' InitVal
变量初值	InitVal	→	Exp | '{' [InitVal { ',' InitVal }] '}'
函数定义	FuncDef	→	FuncType **Ident** '(' [FuncFParams] ')' Block
函数类型	FuncType	→	**'void'** | **'int'** | **'float'**
函数形参表	FuncFParams	→	FuncFParam { ',' FuncFParam }
函数形参	FuncFParam	→	BType **Ident** ['[' ']' { '[' Exp ']' }]
语句块	Block	→	'{' { BlockItem } '}'
语句块项	BlockItem	→	Decl | Stmt
语句	Stmt	→	LVal '=' Exp ';' | [Exp] ';' | Block
			'if' '(Cond ')' Stmt [**'else'** Stmt]
			'while' '(' Cond ')' Stmt
			| **'break'** ';' **'continue'** ';'
			| **'return'** [Exp] ';'
表达式	Exp	→	AddExp 注：SysY 表达式是 int/float 型

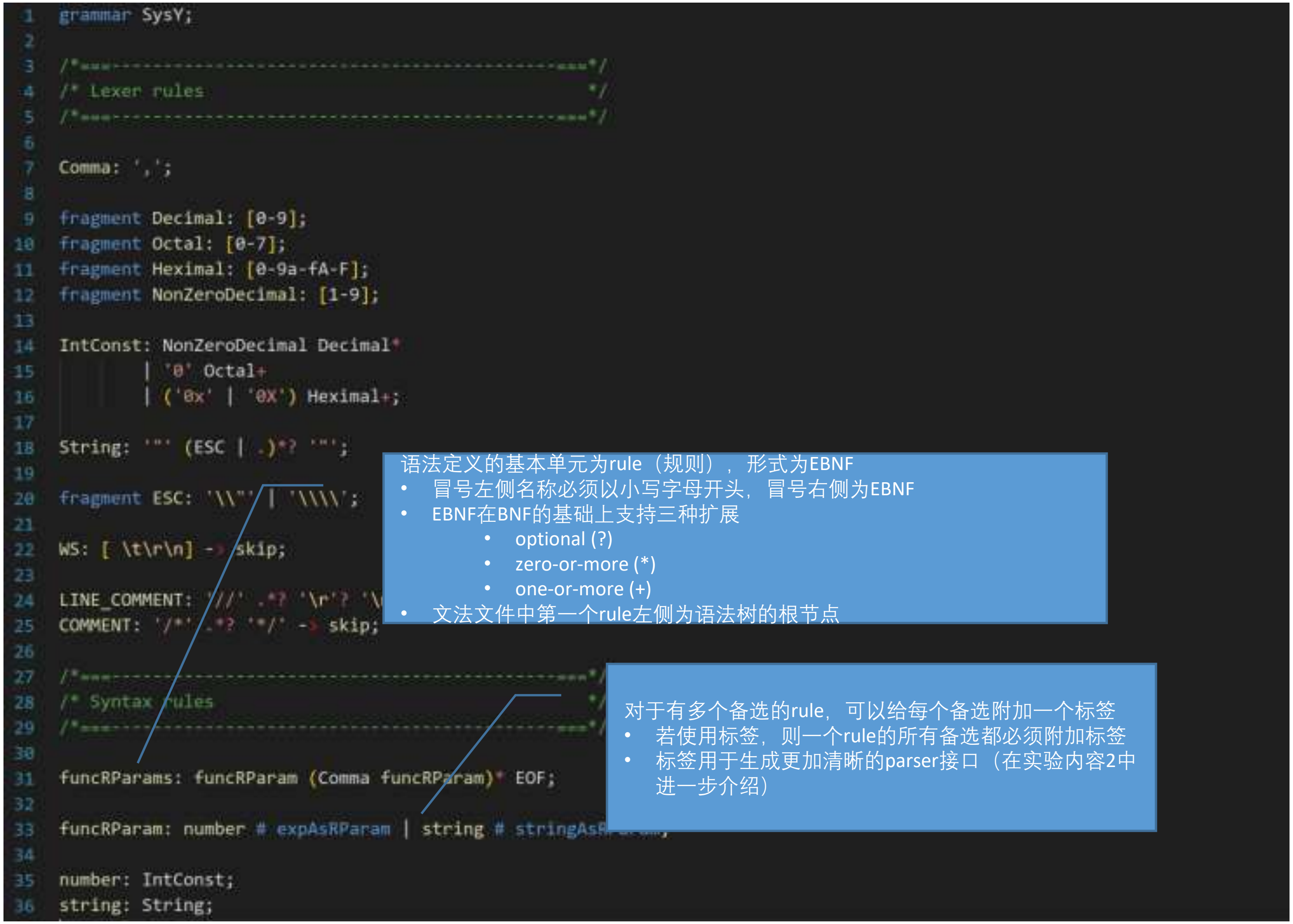


表达式

条件表达式	Cond	→	LOrExp
左值表达式	LVal	→	**Ident** {'[' Exp ']'}
基本表达式	PrimaryExp	→	'(' Exp ')' | LVal | Number
数值	Number	→	**IntConst** | **floatConst**
一元表达式	UnaryExp	→	PrimaryExp | **Ident** '(' [FuncRParams] ')'
			| UnaryOp UnaryExp
单目运算符	UnaryOp	→	'+' | '−' | '!' 注：'!'仅出现在条件表达式中
函数实参表	FuncRParams	→	Exp { ',' Exp }
乘除模表达式	MulExp	→	UnaryExp | MulExp ('*' | '/' | '%') UnaryExp
加减表达式	AddExp	→	MulExp | AddExp ('+' | '−') MulExp
关系表达式	RelExp	→	AddExp | RelExp ('<' | '>' | '<=' | '>=') AddExp
相等性表达式	EqExp	→	RelExp | EqExp ('==' | '!=') RelExp
逻辑与表达式	LAndExp	→	EqExp | LAndExp '&&' EqExp
逻辑或表达式	LOrExp	→	LAndExp | LOrExp '||' LAndExp
常量表达式	ConstExp	→	AddExp 注：使用的 Ident 必须是常量

Token：
粗体字、运算符与标点符号

SysY语法定义



```
grammar SysY;

/*===-------------------------------------------===*/
/* Lexer rules                                    */
/*===-------------------------------------------===*/

Comma: ',';

fragment Decimal: [0-9];
fragment Octal: [0-7];
fragment Heximal: [0-9a-fA-F];
fragment NonZeroDecimal: [1-9];

IntConst: NonZeroDecimal Decimal*
        | '0' Octal+
        | ('0x' | '0X') Heximal+;

String: '"' (ESC | .)*? '"';

fragment ESC: '\\"' | '\\\\';

WS: [ \t\r\n] -> skip;

LINE_COMMENT: '//' .*? '\r'? '\
COMMENT: '/*' .*? '*/' -> skip;

/*===-------------------------------------------===*/
/* Syntax rules                                   */
/*===-------------------------------------------===*/

funcRParams: funcRParam (Comma funcRParam)* EOF;

funcRParam: number # expAsRParam | string # stringAsR

number: IntConst;
string: String;
```

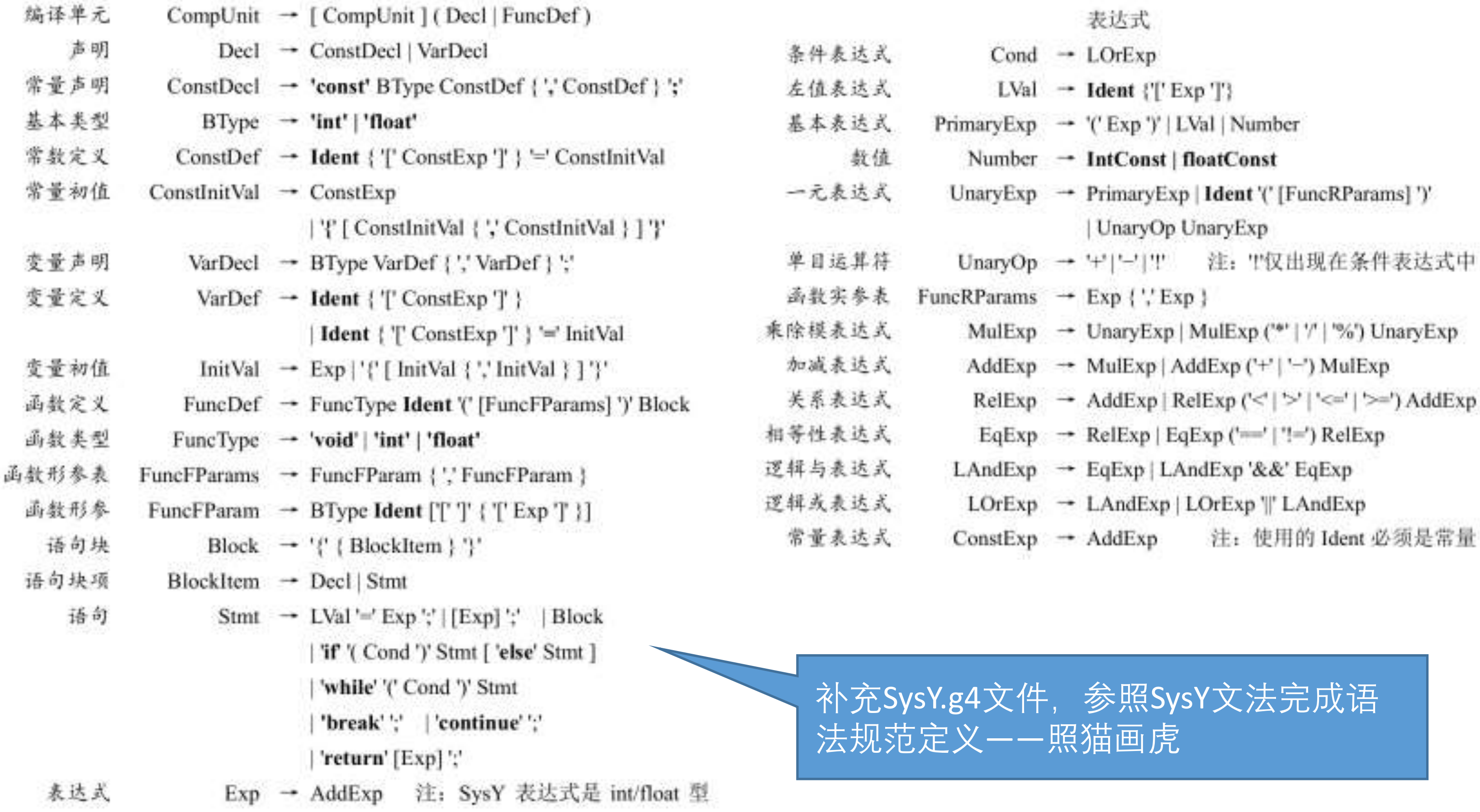
语法定义的基本单元为rule（规则），形式为EBNF

- 冒号左侧名称必须以小写字母开头，冒号右侧为EBNF
- EBNF在BNF的基础上支持三种扩展
  - optional (?)
  - zero-or-more (*)
  - one-or-more (+)
- 文法文件中第一个rule左侧为语法树的根节点

对于有多个备选的rule，可以给每个备选附加一个标签

- 若使用标签，则一个rule的所有备选都必须附加标签
- 标签用于生成更加清晰的parser接口（在实验内容2中进一步介绍）

SysY语法规范



编译单元	CompUnit	→	[CompUnit] (Decl | FuncDef)
声明	Decl	→	ConstDecl | VarDecl
常量声明	ConstDecl	→	**'const'** BType ConstDef { ',' ConstDef } ';'
基本类型	BType	→	**'int'** | **'float'**
常数定义	ConstDef	→	**Ident** { '[' ConstExp ']' } '=' ConstInitVal
常量初值	ConstInitVal	→	ConstExp
			| '{' [ConstInitVal { ',' ConstInitVal }] '}'
变量声明	VarDecl	→	BType VarDef { ',' VarDef } ';'
变量定义	VarDef	→	**Ident** { '[' ConstExp ']' }
			| **Ident** { '[' ConstExp ']' } '=' InitVal
变量初值	InitVal	→	Exp | '{' [InitVal { ',' InitVal }] '}'
函数定义	FuncDef	→	FuncType **Ident** '(' [FuncFParams] ')' Block
函数类型	FuncType	→	**'void'** | **'int'** | **'float'**
函数形参表	FuncFParams	→	FuncFParam { ',' FuncFParam }
函数形参	FuncFParam	→	BType **Ident** ['[' ']' { '[' Exp ']' }]
语句块	Block	→	'{' { BlockItem } '}'
语句块项	BlockItem	→	Decl | Stmt
语句	Stmt	→	LVal '=' Exp ';' | [Exp] ';' | Block
			| **'if'** '(Cond ')' Stmt [**'else'** Stmt]
			| **'while'** '(' Cond ')' Stmt
			| **'break'** ';' | **'continue'** ';'
			| **'return'** [Exp] ';'
表达式	Exp	→	AddExp 注：SysY 表达式是 int/float 型

表达式

条件表达式	Cond	→	LOrExp
左值表达式	LVal	→	**Ident** {'[' Exp ']'}
基本表达式	PrimaryExp	→	'(' Exp ')' | LVal | Number
数值	Number	→	**IntConst** | **floatConst**
一元表达式	UnaryExp	→	PrimaryExp | **Ident** '(' [FuncRParams] ')'
			| UnaryOp UnaryExp
单目运算符	UnaryOp	→	'+' | '−' | '!' 注：'!'仅出现在条件表达式中
函数实参表	FuncRParams	→	Exp { ',' Exp }
乘除模表达式	MulExp	→	UnaryExp | MulExp ('*' | '/' | '%') UnaryExp
加减表达式	AddExp	→	MulExp | AddExp ('+' | '−') MulExp
关系表达式	RelExp	→	AddExp | RelExp ('<' | '>' | '<=' | '>=') AddExp
相等性表达式	EqExp	→	RelExp | EqExp ('==' | '!=') RelExp
逻辑与表达式	LAndExp	→	EqExp | LAndExp '&&' EqExp
逻辑或表达式	LOrExp	→	LAndExp | LOrExp '||' LAndExp
常量表达式	ConstExp	→	AddExp 注：使用的 Ident 必须是常量

补充SysY.g4文件，参照SysY文法完成语法规范定义——照猫画虎

实验内容2

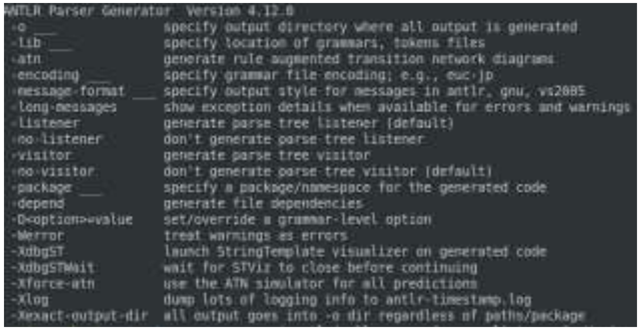
使用ANTLR生成SysY词法/语法分析器

ANTLR使用方法

- 正确设置antlr的运行时环境
  - export CLASSPATH=/path/to/antlr-4.12.0-complete.jar
  - alias antlr4='java -Xmx500M -cp "/path/to/antlr-4.12.0-complete.jar" org.antlr.v4.Tool'
- 运行ANTLR4

  目标语言为C++ 不生成Listener 生成Visitor

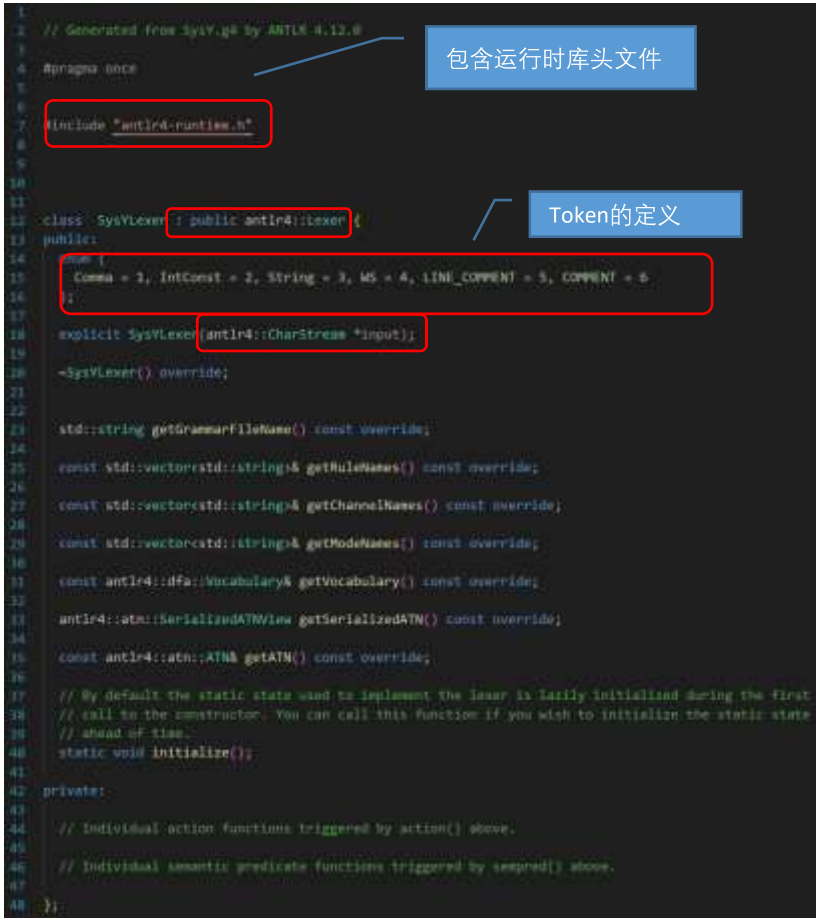
  - antlr4 -Dlanguage=Cpp -no-listener -visitor SysY.g4
- 在当前工作目录生成以下文件
  - SysYLexer.h/SysYLexer.cpp 词法分析器
  - SysYParser.h/SysYParser.cpp 语法分析器
  - SysYVisitor.h/SysYVisitor.cpp Visitor虚基类
  - SysYBaseVisitor.h/SysYBaseVisitor.cpp Visitor基类



```
ANTLR Parser Generator  Version 4.12.0
 -o ___              specify output directory where all output is generated
 -lib ___            specify location of grammars, tokens files
 -atn                generate rule augmented transition network diagrams
 -encoding ___       specify grammar file encoding; e.g., euc-jp
 -message-format ___ specify output style for messages in antlr, gnu, vs2005
 -long-messages      show exception details when available for errors and warnings
 -listener           generate parse tree listener (default)
 -no-listener        don't generate parse tree listener
 -visitor            generate parse tree visitor
 -no-visitor         don't generate parse tree visitor (default)
 -package ___        specify a package/namespace for the generated code
 -depend             generate file dependencies
 -D<option>=value    set/override a grammar-level option
 -Werror             treat warnings as errors
 -XdbgST             launch StringTemplate visualizer on generated code
 -XdbgSTWait         wait for STViz to close before continuing
 -Xforce-atn         use the ATN simulator for all predictions
 -Xlog               dump lots of logging info to antlr-timestamp.log
 -Xexact-output-dir  all output goes into -o dir regardless of paths/package
```

不使用任何参数运行antlr可查看帮助

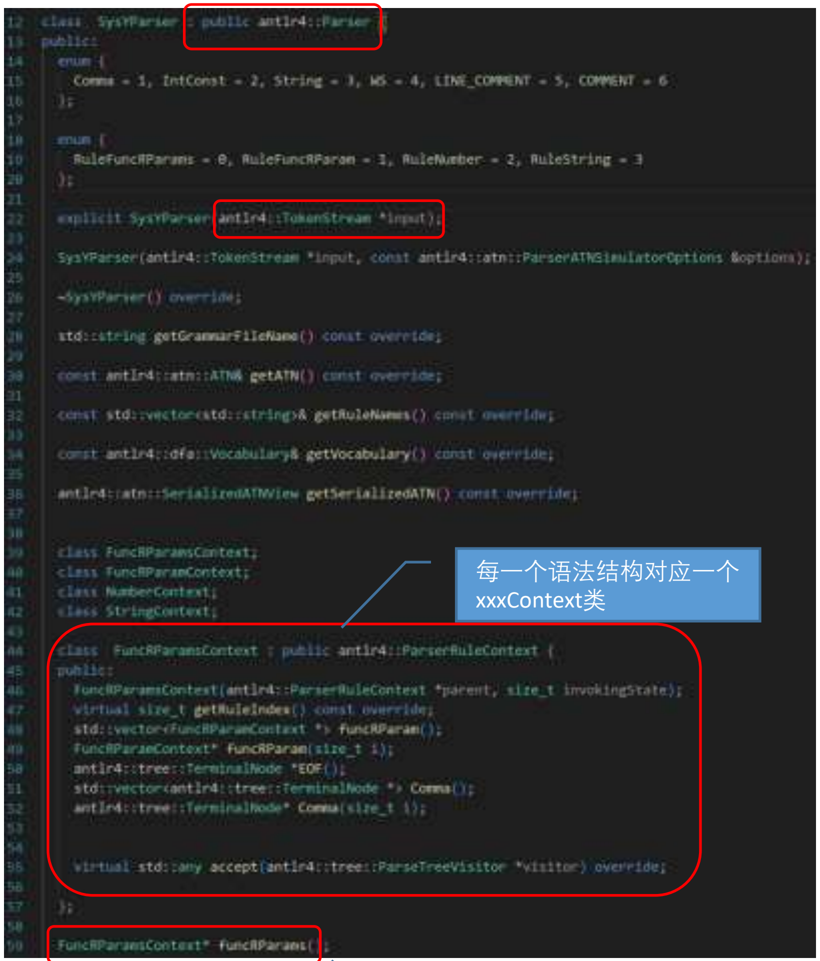
SysYLexer.h/SysYParser.h概览



```cpp
// Generated from SysY.g4 by ANTLR 4.12.0

#pragma once

#include "antlr4-runtime.h"


class  SysYLexer : public antlr4::Lexer {
public:
  enum {
    Comma = 1, IntConst = 2, String = 3, WS = 4, LINE_COMMENT = 5, COMMENT = 6
  };

  explicit SysYLexer(antlr4::CharStream *input);

  ~SysYLexer() override;


  std::string getGrammarFileName() const override;

  const std::vector<std::string>& getRuleNames() const override;

  const std::vector<std::string>& getChannelNames() const override;

  const std::vector<std::string>& getModeNames() const override;

  const antlr4::dfa::Vocabulary& getVocabulary() const override;

  antlr4::atn::SerializedATNView getSerializedATN() const override;

  const antlr4::atn::ATN& getATN() const override;

  // By default the static state used to implement the lexer is lazily initialized during the first
  // call to the constructor. You can call this function if you wish to initialize the static state
  // ahead of time.
  static void initialize();

private:

  // Individual action functions triggered by action() above.

  // Individual semantic predicate functions triggered by sempred() above.

};
```

包含运行时库头文件

Token的定义

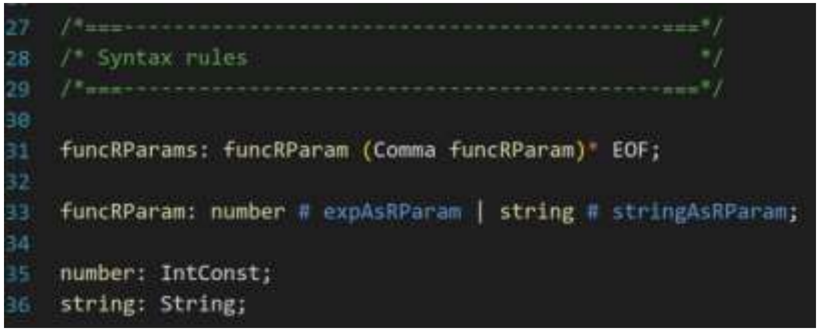


```cpp
class  SysYParser : public antlr4::Parser {
public:
  enum {
    Comma = 1, IntConst = 2, String = 3, WS = 4, LINE_COMMENT = 5, COMMENT = 6
  };

  enum {
    RuleFuncRParams = 0, RuleFuncRParam = 1, RuleNumber = 2, RuleString = 3
  };

  explicit SysYParser(antlr4::TokenStream *input);

  SysYParser(antlr4::TokenStream *input, const antlr4::atn::ParserATNSimulatorOptions &options);

  ~SysYParser() override;

  std::string getGrammarFileName() const override;

  const antlr4::atn::ATN& getATN() const override;

  const std::vector<std::string>& getRuleNames() const override;

  const antlr4::dfa::Vocabulary& getVocabulary() const override;

  antlr4::atn::SerializedATNView getSerializedATN() const override;


  class FuncRParamsContext;
  class FuncRParamContext;
  class NumberContext;
  class StringContext;

  class  FuncRParamsContext : public antlr4::ParserRuleContext {
  public:
    FuncRParamsContext(antlr4::ParserRuleContext *parent, size_t invokingState);
    virtual size_t getRuleIndex() const override;
    std::vector<FuncRParamContext *> funcRParam();
    FuncRParamContext* funcRParam(size_t i);
    antlr4::tree::TerminalNode *EOF();
    std::vector<antlr4::tree::TerminalNode *> Comma();
    antlr4::tree::TerminalNode* Comma(size_t i);


    virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;

  };

  FuncRParamsContext* funcRParams();
```

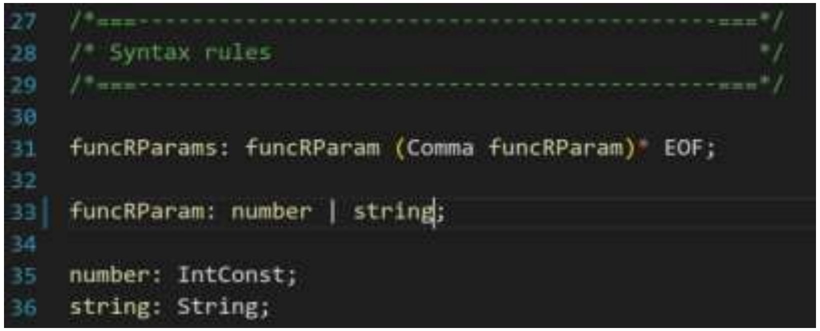
每一个语法结构对应一个xxxContext类

Parser的入口，与文法文件中的root规则同名

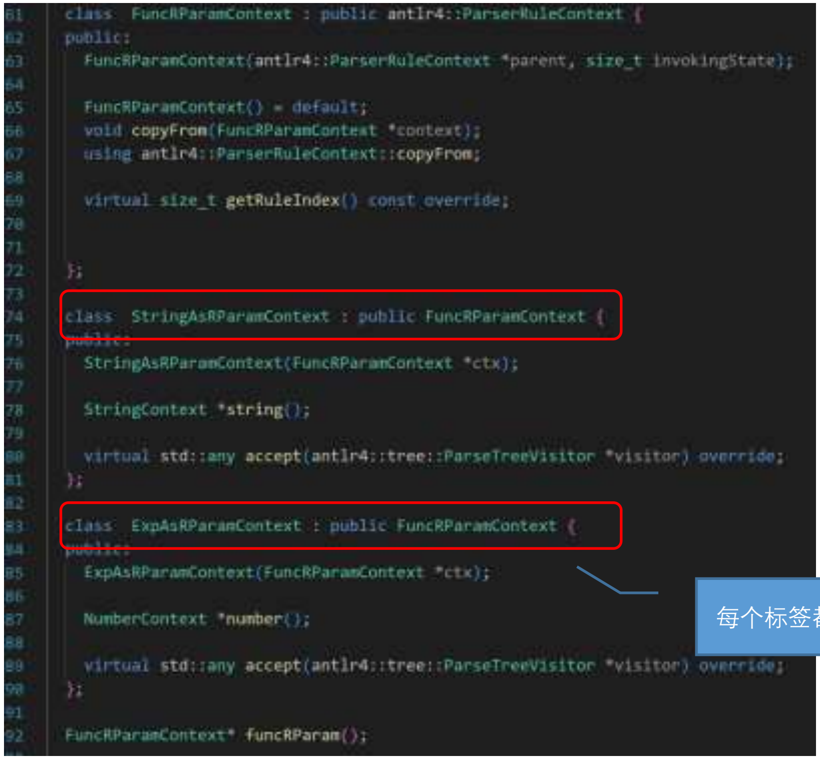
带标签的语法规则



```
/*===-------------------------------------------------===*/
/* Syntax rules                                          */
/*===-------------------------------------------------===*/

funcRParams: funcRParam (Comma funcRParam)* EOF;

funcRParam: number # expAsRParam | string # stringAsRParam;

number: IntConst;
string: String;
```

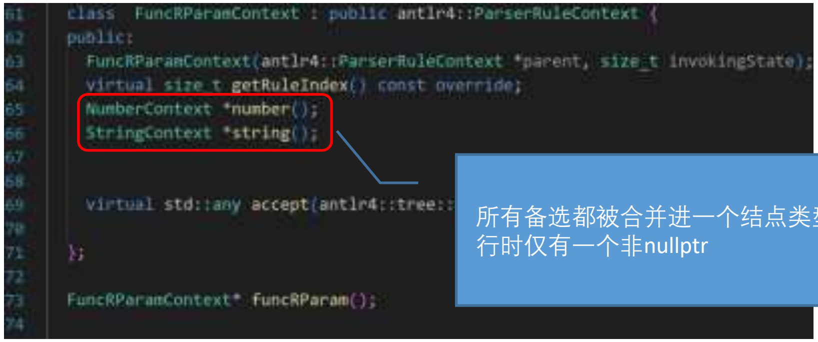


```
/*===-------------------------------------------------===*/
/* Syntax rules                                          */
/*===-------------------------------------------------===*/

funcRParams: funcRParam (Comma funcRParam)* EOF;

funcRParam: number | string;

number: IntConst;
string: String;
```



```
class  FuncRParamContext : public antlr4::ParserRuleContext {
public:
  FuncRParamContext(antlr4::ParserRuleContext *parent, size_t invokingState);

  FuncRParamContext() = default;
  void copyFrom(FuncRParamContext *context);
  using antlr4::ParserRuleContext::copyFrom;

  virtual size_t getRuleIndex() const override;


};

class  StringAsRParamContext : public FuncRParamContext {
public:
  StringAsRParamContext(FuncRParamContext *ctx);

  StringContext *string();

  virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
};

class  ExpAsRParamContext : public FuncRParamContext {
public:
  ExpAsRParamContext(FuncRParamContext *ctx);

  NumberContext *number();

  virtual std::any accept(antlr4::tree::ParseTreeVisitor *visitor) override;
};

FuncRParamContext* funcRParam();
```

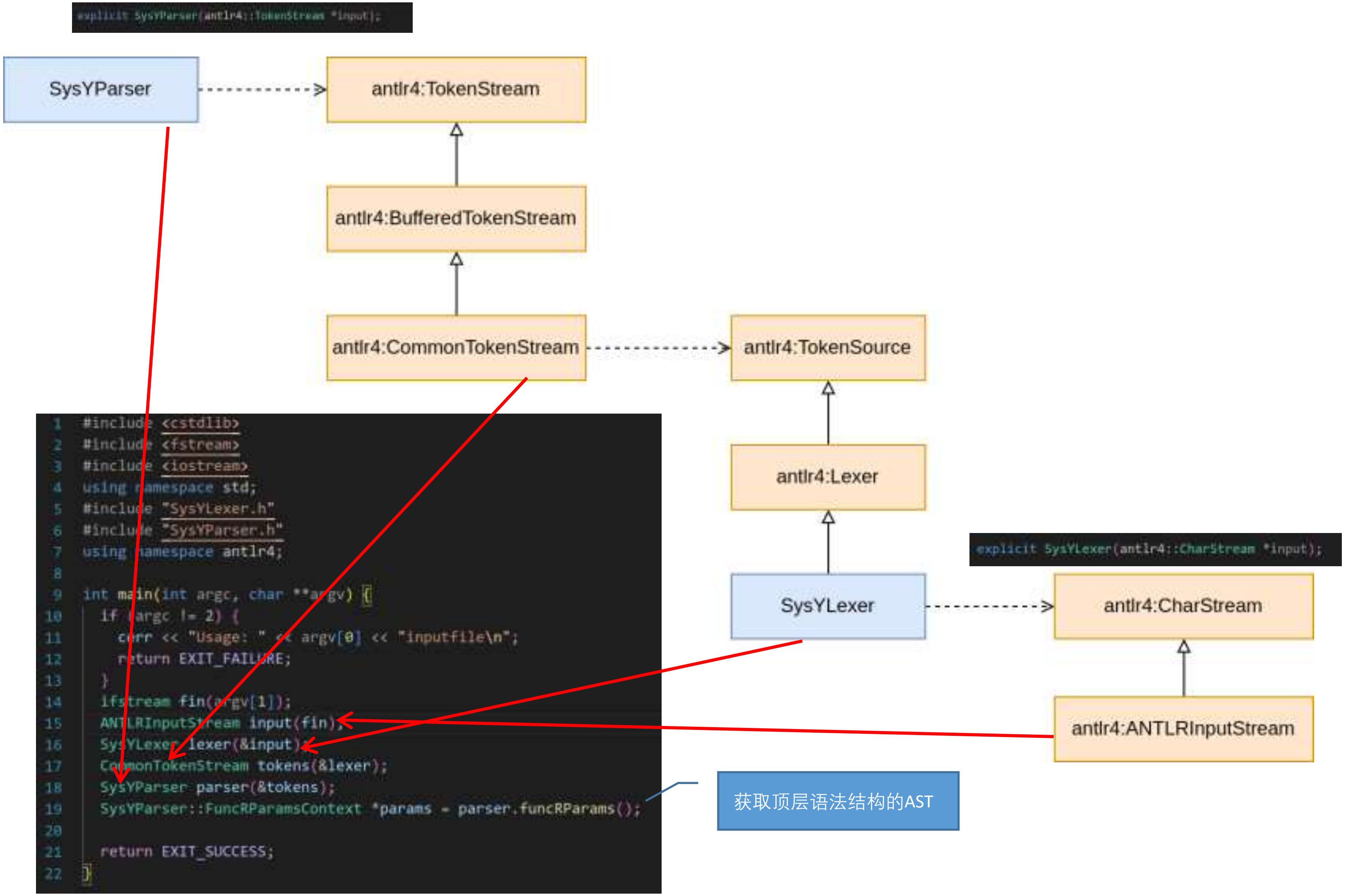
每个标签都会生成一个AST节点类型



```
class  FuncRParamContext : public antlr4::ParserRuleContext {
public:
  FuncRParamContext(antlr4::ParserRuleContext *parent, size_t invokingState);
  virtual size_t getRuleIndex() const override;
  NumberContext *number();
  StringContext *string();


  virtual std::any accept(antlr4::tree::

};

FuncRParamContext* funcRParam();
```

所有备选都被合并进一个结点类型，运行时仅有一个非nullptr

如何构造SysYParser对象?



```
explicit SysYParser(antlr4::TokenStream *input);
```

```
explicit SysYLexer(antlr4::CharStream *input);
```

```
#include <cstdlib>
#include <fstream>
#include <iostream>
using namespace std;
#include "SysYLexer.h"
#include "SysYParser.h"
using namespace antlr4;

int main(int argc, char **argv) {
  if (argc != 2) {
    cerr << "Usage: " << argv[0] << "inputfile\n";
    return EXIT_FAILURE;
  }
  ifstream fin(argv[1]);
  ANTLRInputStream input(fin);
  SysYLexer lexer(&input);
  CommonTokenStream tokens(&lexer);
  SysYParser parser(&tokens);
  SysYParser::FuncRParamsContext *params = parser.funcRParams();

  return EXIT_SUCCESS;
}
```

获取顶层语法结构的AST

实验内容3

基于AST信息输出原始程序

AST Visitor

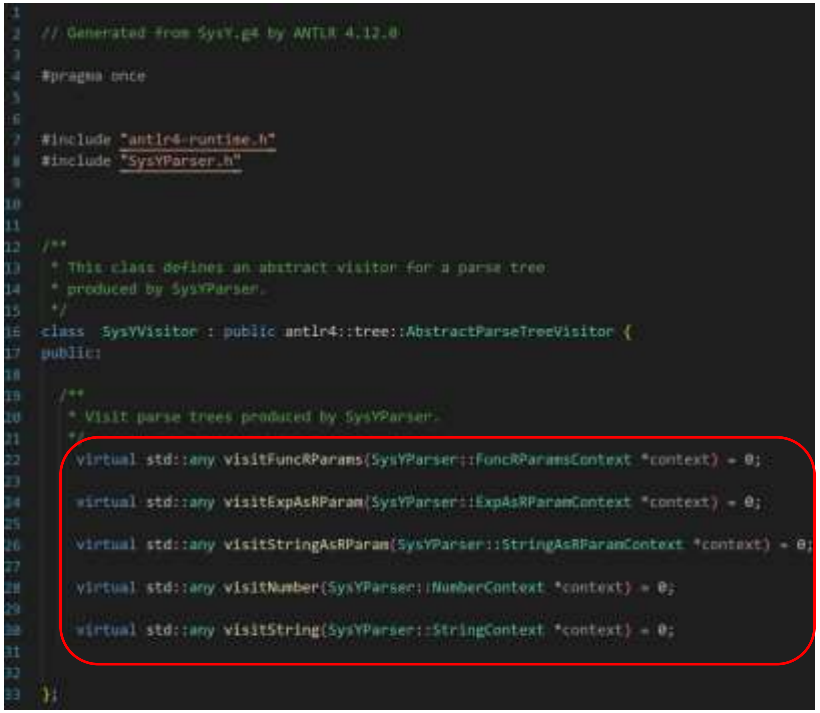
- ANTLR提供了三种方法使用AST
  - 语法制导翻译
  - visitor
  - listener

本实验要求使用visitor

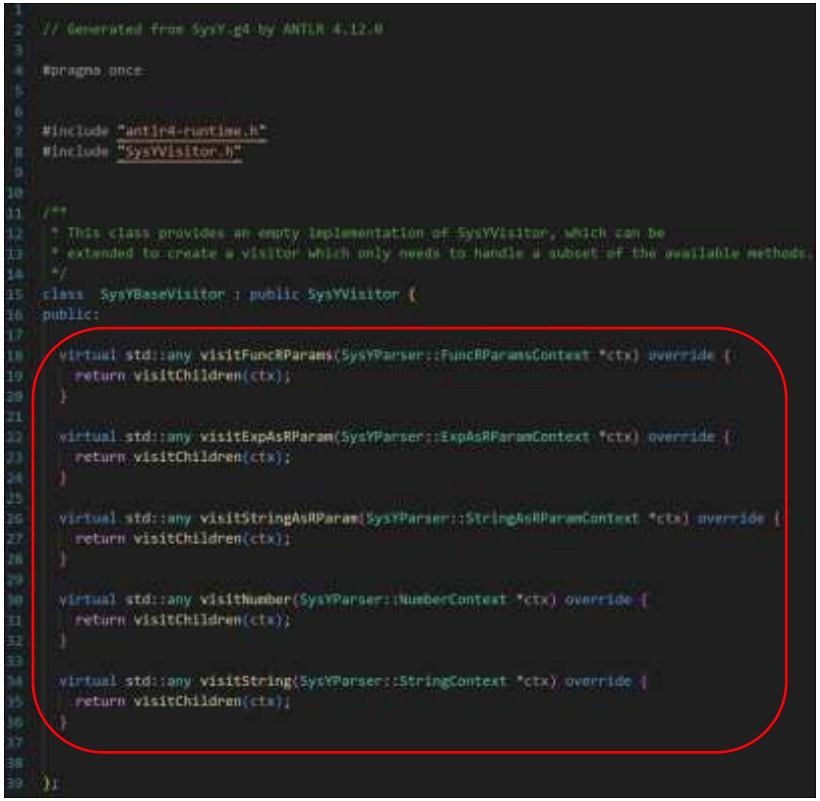
- 回忆：使用-visitor参数令ANTLR生成Visitor类
  - SysYVisitor.h/SysYVisitor.cpp
  - SysYBaseVisitor.h/SysYBaseVisitor.cpp

AST Visitor

类型检查、中间代码生成等过程均可通过继承SysYBaseVisitor实现



```cpp

// Generated from SysY.g4 by ANTLR 4.12.0

#pragma once


#include "antlr4-runtime.h"
#include "SysYParser.h"



/**
 * This class defines an abstract visitor for a parse tree
 * produced by SysYParser.
 */
class  SysYVisitor : public antlr4::tree::AbstractParseTreeVisitor {
public:

  /**
   * Visit parse trees produced by SysYParser.
   */
    virtual std::any visitFuncRParams(SysYParser::FuncRParamsContext *context) = 0;

    virtual std::any visitExpAsRParam(SysYParser::ExpAsRParamContext *context) = 0;

    virtual std::any visitStringAsRParam(SysYParser::StringAsRParamContext *context) = 0;

    virtual std::any visitNumber(SysYParser::NumberContext *context) = 0;

    virtual std::any visitString(SysYParser::StringContext *context) = 0;


};
```

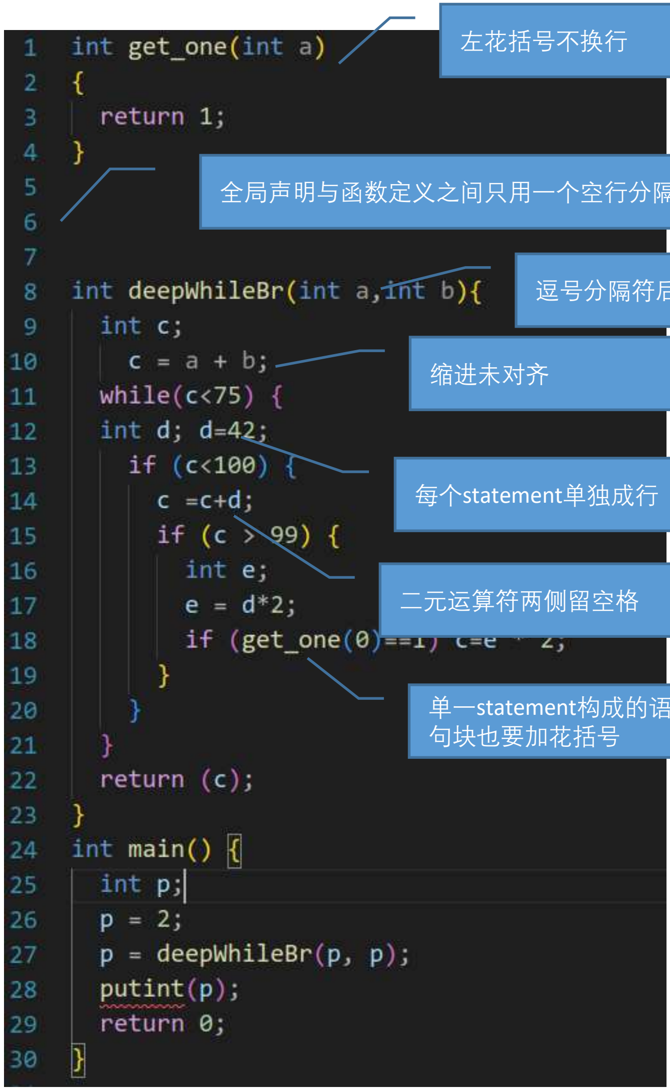


```cpp

// Generated from SysY.g4 by ANTLR 4.12.0

#pragma once


#include "antlr4-runtime.h"
#include "SysYVisitor.h"


/**
 * This class provides an empty implementation of SysYVisitor, which can be
 * extended to create a visitor which only needs to handle a subset of the available methods.
 */
class  SysYBaseVisitor : public SysYVisitor {
public:

  virtual std::any visitFuncRParams(SysYParser::FuncRParamsContext *ctx) override {
    return visitChildren(ctx);
  }

  virtual std::any visitExpAsRParam(SysYParser::ExpAsRParamContext *ctx) override {
    return visitChildren(ctx);
  }

  virtual std::any visitStringAsRParam(SysYParser::StringAsRParamContext *ctx) override {
    return visitChildren(ctx);
  }

  virtual std::any visitNumber(SysYParser::NumberContext *ctx) override {
    return visitChildren(ctx);
  }

  virtual std::any visitString(SysYParser::StringContext *ctx) override {
    return visitChildren(ctx);
  }


};
```

Visitor对每一种AST结点类型均定义了一个访问方法visitXXX

- SysYVisitor是一个虚基类，仅定义接口
- SysYBaseVisitor提供了SysYVisitor的默认实现
  - 每个结点访问方法仅递归向下访问所有子节点
  - 用户可继承SysYBaseVisitor，覆盖部分结点的访问方法

SysY语言格式化器



```
int get_one(int a)
{
  return 1;
}



int deepWhileBr(int a,int b){
  int c;
    c = a + b;
  while(c<75) {
  int d; d=42;
    if (c<100) {
      c =c+d;
      if (c > 99) {
        int e;
        e = d*2;
        if (get_one(0)==1) c=e * 2;
      }
    }
  }
  return (c);
}
int main() {
  int p;
  p = 2;
  p = deepWhileBr(p, p);
  putint(p);
  return 0;
}
```

左花括号不换行

全局声明与函数定义之间只用一个空行分隔

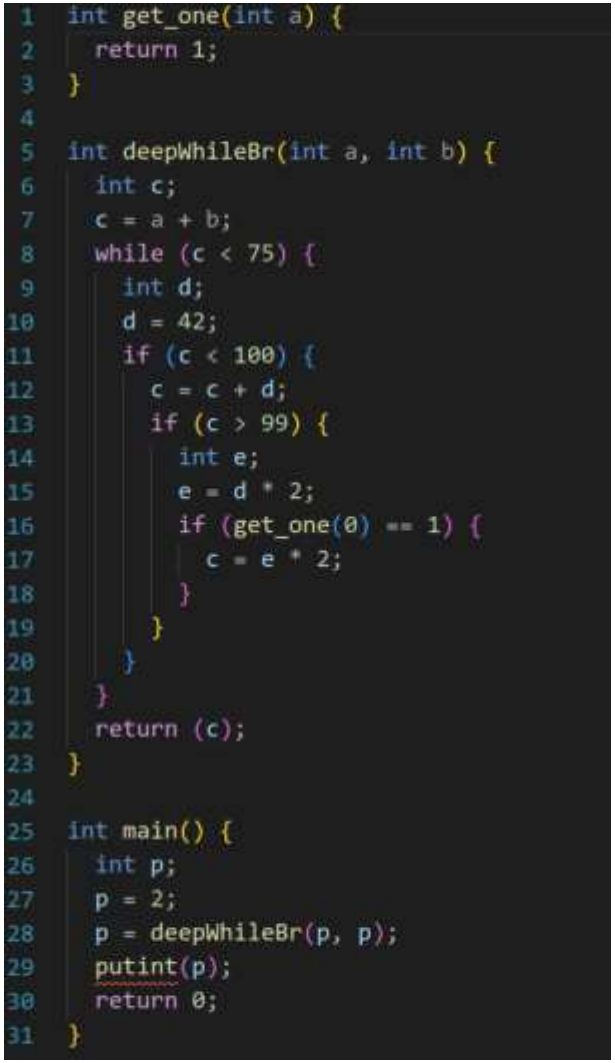
逗号分隔符后有一个空格

缩进未对齐

每个statement单独成行

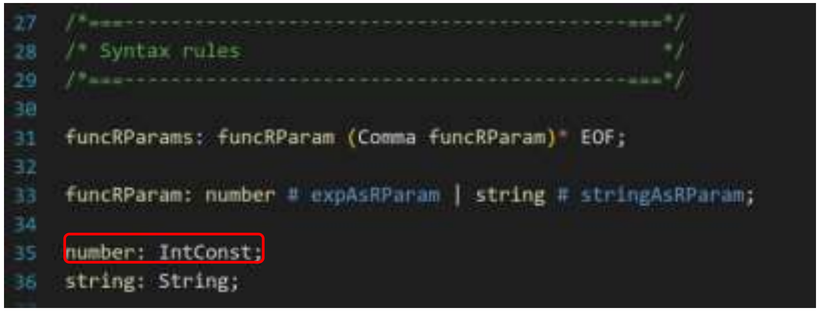
二元运算符两侧留空格

单一statement构成的语句块也要加花括号

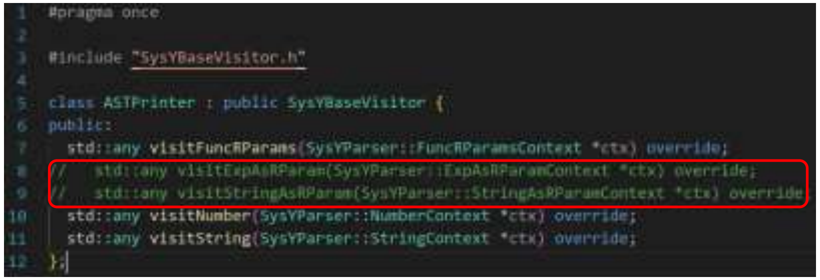


```
int get_one(int a) {
  return 1;
}

int deepWhileBr(int a, int b) {
  int c;
  c = a + b;
  while (c < 75) {
    int d;
    d = 42;
    if (c < 100) {
      c = c + d;
      if (c > 99) {
        int e;
        e = d * 2;
        if (get_one(0) == 1) {
          c = e * 2;
        }
      }
    }
  }
  return (c);
}

int main() {
  int p;
  p = 2;
  p = deepWhileBr(p, p);
  putint(p);
  return 0;
}
```

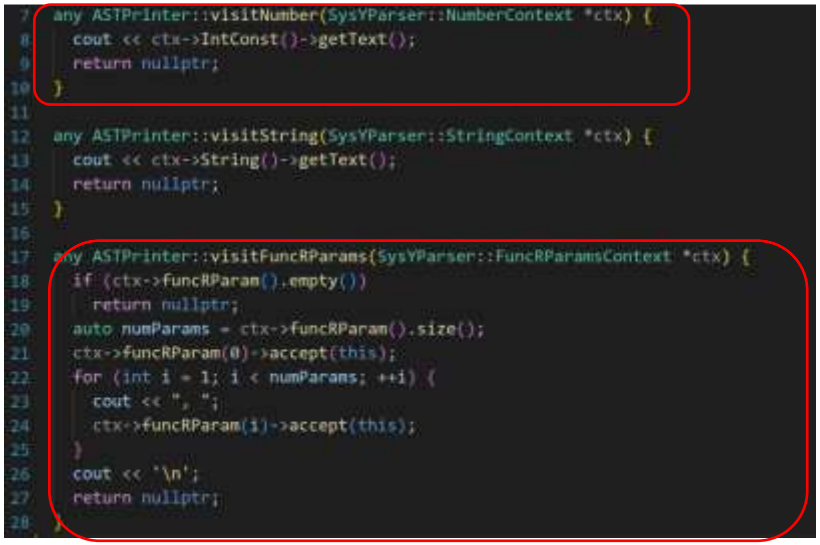
实现思路



```
/*===-------------------------------------------===*/
/* Syntax rules                                    */
/*===-------------------------------------------===*/

funcRParams: funcRParam (Comma funcRParam)* EOF;

funcRParam: number # expAsRParam | string # stringAsRParam;

number: IntConst;
string: String;
```



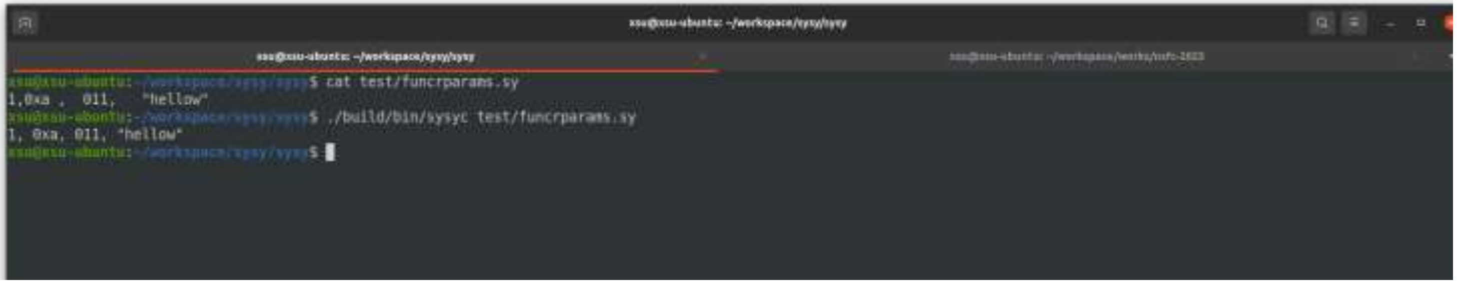
```
#pragma once

#include "SysYBaseVisitor.h"

class ASTPrinter : public SysYBaseVisitor {
public:
  std::any visitFuncRParams(SysYParser::FuncRParamsContext *ctx) override;
//   std::any visitExpAsRParam(SysYParser::ExpAsRParamContext *ctx) override;
//   std::any visitStringAsRParam(SysYParser::StringAsRParamContext *ctx) override
  std::any visitNumber(SysYParser::NumberContext *ctx) override;
  std::any visitString(SysYParser::StringContext *ctx) override;
};
```



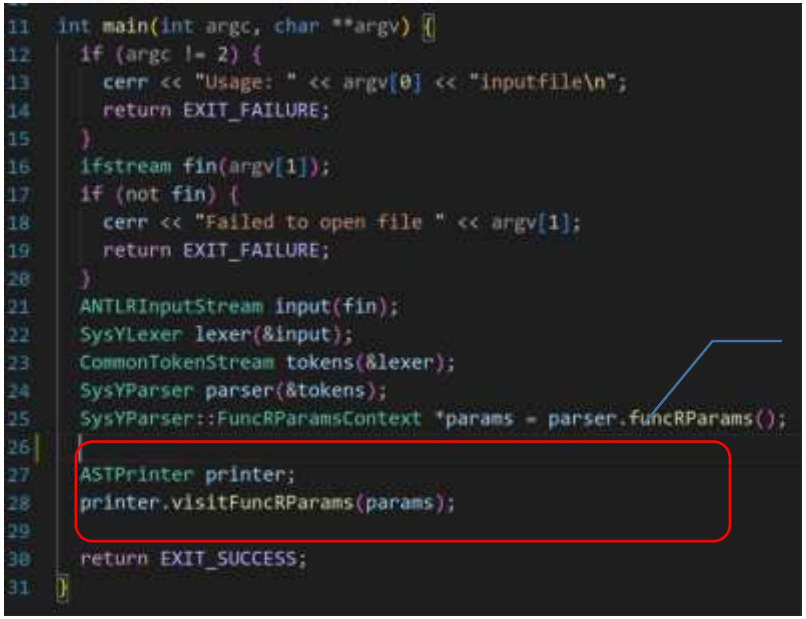
```
any ASTPrinter::visitNumber(SysYParser::NumberContext *ctx) {
  cout << ctx->IntConst()->getText();
  return nullptr;
}

any ASTPrinter::visitString(SysYParser::StringContext *ctx) {
  cout << ctx->String()->getText();
  return nullptr;
}

any ASTPrinter::visitFuncRParams(SysYParser::FuncRParamsContext *ctx) {
  if (ctx->funcRParam().empty())
    return nullptr;
  auto numParams = ctx->funcRParam().size();
  ctx->funcRParam(0)->accept(this);
  for (int i = 1; i < numParams; ++i) {
    cout << ", ";
    ctx->funcRParam(i)->accept(this);
  }
  cout << '\n';
  return nullptr;
}
```

- 覆写（override）SysYBaseVisitor类的方法，从AST结点输出源程序
- 对于number/string节点，直接输出对应字符串
- 对于funcRParam节点，只需要处理其子节点，SysYBaseVisitor的默认实现即可，无需覆写
- 对于funcRParams节点，逐个输出子节点，相邻子节点之间输出“，”

格式化器测试



```
xsu@xsu-ubuntu:~/workspace/sysy/sysy$ cat test/funcrparams.sy
1,0xa ,  011,   "hellow"
xsu@xsu-ubuntu:~/workspace/sysy/sysy$ ./build/bin/sysyc test/funcrparams.sy
1, 0xa, 011, "hellow"
xsu@xsu-ubuntu:~/workspace/sysy/sysy$ 
```



```
int main(int argc, char **argv) {
  if (argc != 2) {
    cerr << "Usage: " << argv[0] << "inputfile\n";
    return EXIT_FAILURE;
  }
  ifstream fin(argv[1]);
  if (not fin) {
    cerr << "Failed to open file " << argv[1];
    return EXIT_FAILURE;
  }
  ANTLRInputStream input(fin);
  SysYLexer lexer(&input);
  CommonTokenStream tokens(&lexer);
  SysYParser parser(&tokens);
  SysYParser::FuncRParamsContext *params = parser.funcRParams();

  ASTPrinter printer;
  printer.visitFuncRParams(params);

  return EXIT_SUCCESS;
}
```

在获得AST后，使用ASTPrinter类对AST进行处理，输出格式化后的程序

实验内容

- 定义SysY语言的词法/语法规范
- 使用ANTLR工具生成SysY语言的词法/语法分析器
- 实现SysY语言格式化器（进阶内容）

Let’s Go!