编译原理研讨课

崔慧敏
cuihm@ict.ac.cn
http://www.carch.ac.cn/~huimin/main.html
助教：李奕瑾, liyijin@ict.ac.cn
李帅江， lishuaijiang19b@ict.ac.cn

提纲

- 编译原理研讨课实验
- 如何构造一个编译器?
  - 工业级编译器Clang/LLVM
  - 课程作业级别的编译器
  - 编译整体的工作流程
- 作业时间表：TBA
  - PR001，今天~待定
  - PR002，根据课程进度决定
  - PR003，根据课程进度决定
- ANTLR详细介绍+demo介绍

编译原理研讨课课程实验

- 实验任务：

  构思并实现一个端到端的编译系统

  - 推荐C++11实现，课程提供支持
  - 其他开发语言提供有限支持

  能够在x86 Linux OS上运行，将CACT语言的源程序编译为RISC-V汇编

  目标是掌握编译器的完整工作流程，及相关优化技术

- 评测：

  功能性：具有编译器能力，支持错误处理

  正确性：输入测试程序源码，生成功能正确的RISC-V汇编

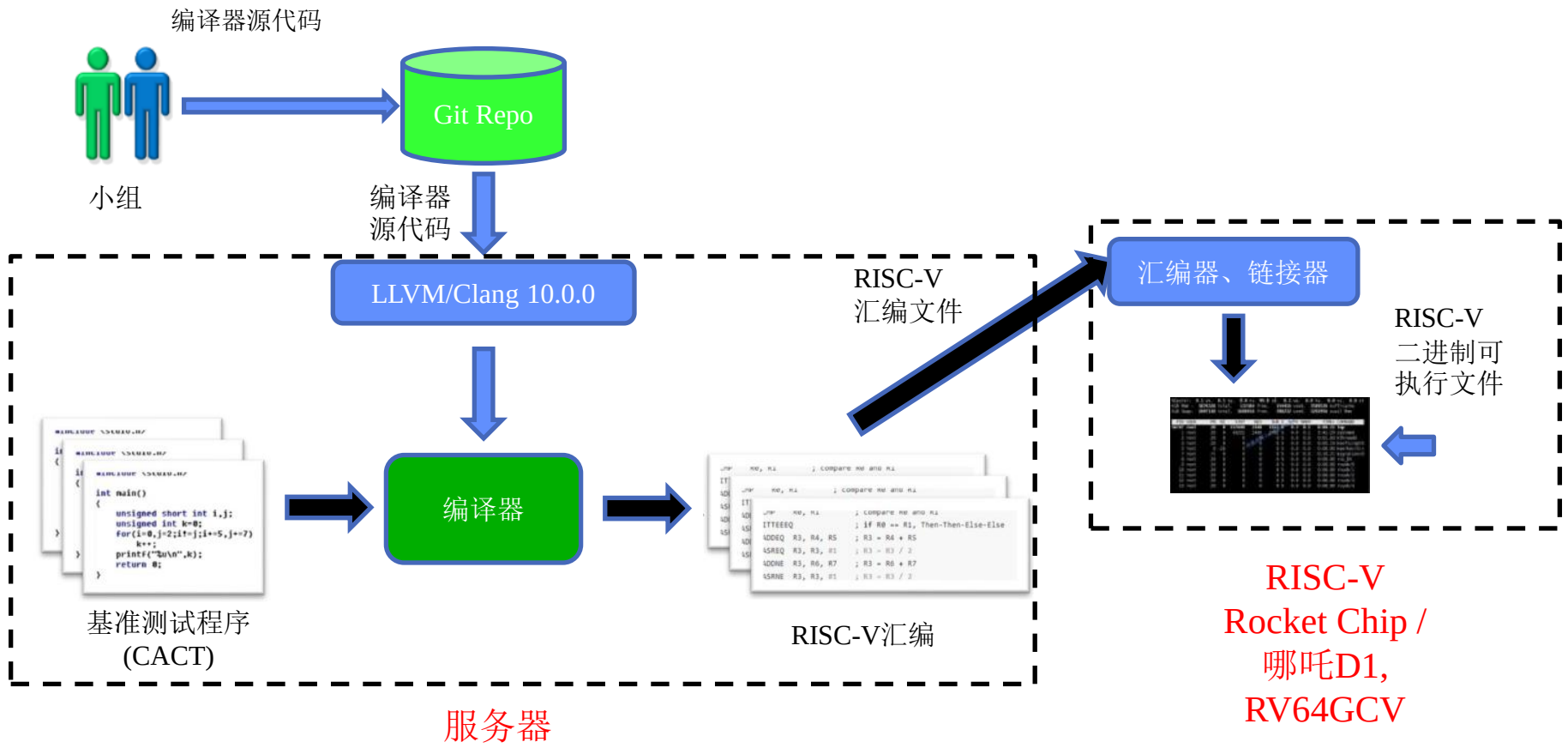
  性能：生成的汇编代码的执行时间

课程要求

- 支持语言： 推荐C/C++，使用其他语言开发课程提供有限支持
- 编译环境： LLVM/Clang 10.0.0

  clang++ –std=c11 –O2 -lm
- 需从头构造编译系统，不得直接使用现有开源编译器及框架的源代码及裁剪
- 推荐使用ANTLR工具辅助生成词法、语法分析代码，但这不是必须要求，手写递归下降或者Lex/Yacc等工具，提供有限支持
- 各小组可自行决定编译器的体系结构、前后端设计等细节
- 如果使用他人源码，必须在设计文档和源程序文件头部予以明确说明

评测过程


编译器源代码
Git Repo
小组
编译器
源代码
LLVM/Clang 10.0.0
RISC-V
汇编文件
汇编器、链接器
RISC-V
二进制可
执行文件
编译器
基准测试程序
(CACT)
RISC-V汇编
服务器
RISC-V
Rocket Chip /
哪吒D1,
RV64GCV

实验说明

- PR001：词法分析和语法分析
  - 推荐使用ANTLR工具
  - 允许自行编写手工下降的代码
- PR002：语义分析
  - 生成AST，进行语义检查
- PR003：代码生成及优化
  - AST到RISC-V指令集
  - 实现一些编译优化

目标语言： CACT

- 语言特点：CACT
  - C语言子集 （*.cact）,以及一点扩展
  - 只包含一个源文件
    - 无头文件，无include
  - 主函数定义main
  - 全局变量声明、常量声明、函数声明、定义
  - 不允许任何的类型转换
    - A = 3.0; //编译错误
  - 类型系统：
    - 支持 int 、bool、float、double
    - 一维数组，编译器常量
  - 扩展：数组的逐元素运算
    - int A[100]; C = A + B ;
- 语言特性：
  - 赋值、表达式、 if、 while、 break、 continue
  - 语句块支持若干变量声明和语句
  - 支持基本算术运算、关系运算、逻辑运算
  - 算符优先级和结合性与C语言保持一致

```
float n;
int whileIf() {
  int a, b[10];
  int c[10], d[10];
  while (a < 100) {
    if (a == 5)
      b = c + d;
    else
      b = c * d;
    a = a + 1;
  }
  return (b[0]);
}
int main(){
  return (whileIf());
}
```

目标语言： CACT

- 语法描述：扩展的Backus范式（EBNF）

语句	Stmt	→ LVal '=' Exp ';' | [Exp] ';' | Block | **'if'** '(' Cond ')' Stmt [**'else'** Stmt] | **'while'** '(' Cond ')' Stmt | **'break'** ';' | **'continue'** ';' | **'return'** [Exp] ';'
表达式	Exp	→ AddExp | **BoolConst**

```
int whileIf() {
  int a, b[10];
  int c[10], d[10];
  while (a < 100) {
    if (a == 5)
      b = c + d;
    else
      b = c * d;
    a = a + 1;
  }
  return (b[0]);
}
int main(){
  return (whileIf());
}
```

目标语言： CACT

- 输入输出库说明：

  CACT不提供I/O库

  - 如C中stdio.h中的函数，printf等

  CACT提供以下七个输出函数及输入函数：

  - print_bool/int/float/double
  - get_int/float/double
  - 课程提供标准库实现：.a和.so形式
  - CACT用户代码可直接引用这七个函数
  - 编译器自行处理七个函数的符号表等内容

  视情况添加更多的此类函数

语言实例1

- 合法示例:

```
constDecl
    : 'const' bType constDef (',' constDef)* ';'
    ;
```

```
constDef
    : Ident ('[' constExp ']')? '=' constInitVal
    ;
```

```
vardecl
    : bType varDef (',' varDef)* ';'
    ;
```

```
varDef
    : Ident ('[' IntConst ']')?
    | Ident ('[' IntConst ']')? '=' constInitVal
    ;
```

```
const int a = 5;
int b;
int c[1000];
int main(){
    a=10;
    int c;
    c=10;
    return 0;
}
```

语言实例2

- 合法示例:

```
funcDef
    : funcType Ident '(' (funcFParams)? ')' block
    ;
```

```
funcFParams
    : funcFParam (',' funcFParam)*
    ;

funcFParam
    : bType Ident ('[' ']')?
    ;
```

```
funcRParams
    : exp (',' exp)*
    ;
```

```
int a;
int func(int p){
        p = p - 1;
        return p;
}
int main(){
        int b;
        a = 10;
        b = func(a);
        return b;
}
```

语言实例3

- 不合法示例:

```
funcDef
    : funcType Ident '(' (funcFParams)? ')' block
    ;
```

```
funcFParams
    : funcFParam (',' funcFParam)*
    ;

funcFParam
    : bType Ident ('[' ']')?
    ;
```

```
funcRParams
    : exp (',' exp)*
    ;
```

语义错误，语法合法

```
int a;
int func1(int p){
        p = p - 1;
        return p;
}
int main(){
        int b;
        a = 10;
        c = func2(a);
        return b;
}
```

语言实例4

- 合法示例:

```
stmt
    : lVal '=' exp ';'
    | (exp)? ';'
    | block
    | 'if' '(' cond ')' stmt ('else' stmt)?
    | 'while' '(' cond ')' stmt
    | 'break' ';'
    | 'continue' ';'
    | 'return' (exp)? ';'
    ;
```

数组加减

```
int whileIf() {
  int a, b[10];
  int c[10], d[10];
  while (a < 100) {
    if (a == 5)
      b = c + d;
    else
      b = c * d;
    a = a + 1;
  }
  return (b[0]);
}
int main(){
  return (whileIf());
}
```

语言实例5

- 不合法示例:

```
OctalConst
    : '0' OctalDigit+
    ;
```

```
OctalDigit
    : [0-7]
    ;
```

```
HexadecimalConst
    : HexadecimalPrefix HexadecimalDigit+
    ;
```

```
HexadecimalPrefix
    : '0x'
    | '0X'
    ;
```

```
HexadecimalDigit
    : [0-9a-fA-F]
    ;
```

数字定义不合法

```
int main(){
        int i = 1;
        int j = ~1;
        int k = 0129;
        int n = 0x3G;
        return 0;
}
```

语言实例6

- 不合法示例:

```
Ident
    : IdentNondigit [a-zA-Z_0-9]*
    ;
```

```
IdentNondigit
    : [a-zA-Z_]
    ;
```

数字、变量名不合法

```
int main(){
        int i = "ha";
        int 7j = 5;
        return 0;
}
```

语言实例7

- 不合法示例:

```
lVal
    : Ident ('[' exp ']')?
    ;
```

```
vardecl
    : bType varDef (',' varDef)* ';'
    ;
```

```
stmt
    : lVal '=' exp ';'
    | (exp)? ';'
    | block
    | 'if' '(' cond ')' stmt ('else' stmt)?
    | 'while' '(' cond ')' stmt
    | 'break' ';'
    | 'continue' ';'
    | 'return' (exp)? ';'
    ;
```

数组访问、语句不合法

```
int main()
{
        float a[10];
        int i
        a[5,3] = 1.5;
        if(a[1] == 0)
                i = 1
        else
                i = 0;
        return 0
}
```

语言实例8

- 不合法示例:

```
BlockComment
    : '/*' .*? '*/'
    -> skip
    ;

LineComment
    : '//' ~[\r\n]*
    -> skip
    ;
```

嵌套注释不合法

```
int main()
{
        // line comment
        float a[10];
        int i = 3;
        /* block comment */
        /*
        /*a[5,3] = 1.5;*/
        */
        if(a[1] == 0)
                i = 1
        else
                i = 0;
        return 0
}
```

目标体系结构：RISC-V

- RISC-V是一种指令集

  AMD64 (Intel/AMD/VIA + 海光)，CISC

  ARMv8a (ARM + 苹果/高通/)，RISC

  MIPS以及LoongArch（龙芯），RISC

  目前不活跃的：Alpha、Sparc等，RISC居多

  指令集是计算机软件和硬件的最主要的接口

- RISC-V历史：

  开源指令集架构，“*completely open*”

  起源于UC Berkeley， David Patterson，2010

  目前由RISC-V基金会运作

目标体系结构：RISC-V

- RISC-V的模块化
- 四个基本的指令集：

  RV32I，RV32E，RV64I，RV128I

  RV32I保持不变
- 标准扩展：

  I Integer and basic instructions

  M Multiply and divide

  A Atomics

  F IEEE floating point (single precision)

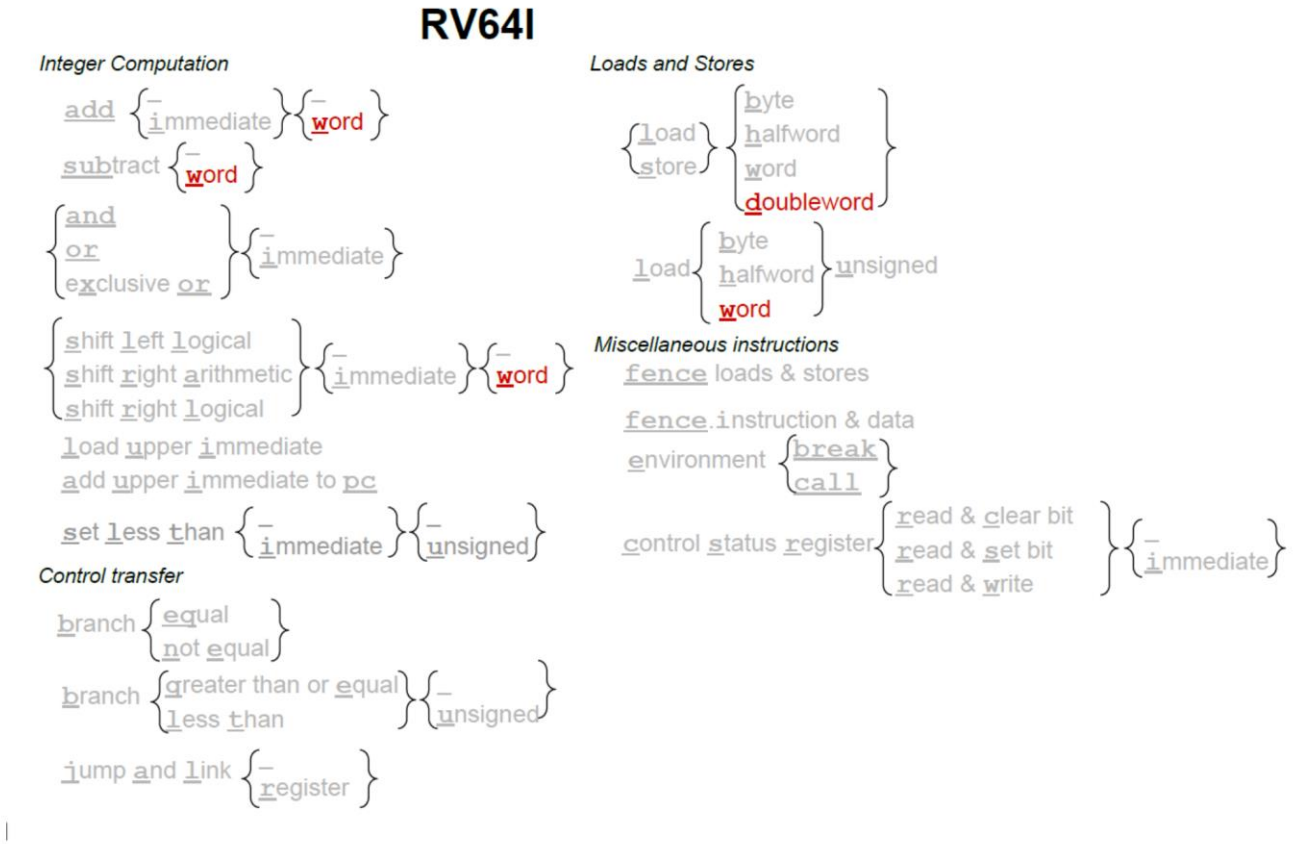
  D IEEE floating point (double precision)

  C Compressed instructions

ISA	Pages	Words
RISC-V	236	76,702
ARM-32	2736	895,032
x86-32	2198	2,186,259

目标体系结构：RISC-V

- CPU: Rocket Chip，指令集：RV64IMAFDC (RV64GC)



目标体系结构：RISC-V

- CPU: Rocket Chip，指令集：RV64IMAFDC (RV64GC)

```
2   int main(){
3       int a = 5;
4       int b = 3;
5       return a * b;
6   }
```

```
21          addi    s0, sp, 32
22          .cfi_def_cfa s0, 0
23          mv      a0, zero
24          sw      a0, -20(s0)
25          addi    a0, zero, 5
26  .Ltmp0:
27          .loc    1 3 9 prologue_end              # ./example.c:3:9
28          sw      a0, -24(s0)
29          addi    a0, zero, 3
30          .loc    1 4 9                           # ./example.c:4:9
31          sw      a0, -28(s0)
32          .loc    1 5 12                          # ./example.c:5:12
33          lw      a0, -24(s0)
34          .loc    1 5 16 is_stmt 0                # ./example.c:5:16
35          lw      a1, -28(s0)
36          .loc    1 5 14                          # ./example.c:5:14
37          mulw    a0, a0, a1
38          .loc    1 5 5                           # ./example.c:5:5
39          ld      s0, 16(sp)
40          ld      ra, 24(sp)
41          addi    sp, sp, 32
42          ret
43  .Ltmp1:
```

评测与打分

- 最终的成绩由三部分组成：
  - 实验报告
  - 测试样例通过率（功能分）
  - 生成的汇编代码的性能（性能分）。
- 测试用例分为两种
  - 功能测试用例：大部分公开用例 + 一些隐藏用例（PR001-PR003均有）
  - 性能测试用例（仅出现在PR003）。
- 公开测试用例与性能测试用例均会放在课程网站和Gitlab上

如何构造一个编译器?

- 编译原理教科书:

  词法分析，语法分析，语义分析，中间代码生成，中间代码优化，代码生成

  文法，乔姆斯基文法分类

  语法制导的翻译

  存储过程

  寄存器分配等等

如何构造一个编译器?

- 工业级编译器：Clang/LLVM:

  词法分析和语法分析以及语义分析：手写的递归下降

  - 支持C/C++/Obj-C/Objc-C++等

  中间代码：LLVM IR

  - 机器无关的中间表示IR
  - SSA
  - 无限寄存器

  目标体系结构:

  - AMD64，RISC-V，ARM等等

如何构造一个编译器?

- 课程实验编译器:

  词法分析和语法分析以及语义分析

  - Option 1: 手写递归下降
  - Option 2: 使用工具自动生成Lexer/Parser
  - 输出抽象语法树AST

  中间表示:

  - Option 1: LLVM IR
  - Option 2: 自行设计简单的三地址中间表示
  - AST到中间表示

  目标体系结构: 中间表示到RISC-V汇编

  - 仅需考虑RISC-V

编译流程实例

- 编译过程:

  源代码 -> 汇编代码 -> 目标文件 -> 可执行文件

  1.c -> 1.s-> 1.o -> a.out

- Clang/LLVM编译过程:

  源代码 -> LLVM IR -> 汇编代码 -> 目标文件 -> 可执行文件

  clang 1.c

ANTLR

- PR001作业的核心
- ANother Tool for Language Recognition
- 词法分析+语法分析

  输入：词法和语法描述（.g4文件）

  输出：词法、语法分析器：Lexer, Parser, 访问接口：Listener/Visitor

  Target：Java, C#, Python2|3, JavaScript, Go, C++, Swift
- ANTLR生成的parser可以自动生成语法树，并提供遍历该语法树的接口

  Listener模式/Visitor模式

ANTLR

```
e : e '*' e
  | e '+' e
  | INT
  ;
```

- LL(*)
- 可处理直接左递归，不可处理间接左递归
- 官方给出了针对很多语言的grammar，可以参考学习

  https://github.com/antlr/grammars-v4
- 应用：语言解析-不止是编译器

  presto使用antlr做底层sql语法解析等

ANTLR - 例子

- 语法文件分为两部分

  Parser部分：小写开头

  Lexer部分：大写开头

编译单元 CompUnit → [CompUnit] (Decl | FuncDef)

声明 Decl → ConstDecl | VarDecl

常量声明 ConstDecl → **'const'** BType ConstDef { ',' ConstDef } **';'**

标识符 identifier → identifier-nondigit
| identifier identifier-nondigit
| identifier digit

```
Ident
    : IdentNondigit [a-zA-Z_0-9]*
    ;

fragment
IdentNondigit
    : [a-zA-Z_]
    ;
```

EBNF范式 词法描述

```
compUnit
    : (decl | funcDef)+ EOF
    ;

decl
    : constDecl          # constDeclDecl
    | vardecl            # varDeclDecl
    ;

constDecl
    : 'const' bType constDef (',' constDef)* ';'
    ;
```

EBNF范式 语法描述

ANTLR - Listener模式

- 默认为Listener模式：`$antlr4 example.g4`

  通过结点监听，进入和退出规则节点时，触发相应方法

  深度优先遍历

- 优点：

  自动调用结点遍历方法，实现简单

  动作代码与文法产生式解耦，利于文法产生式的重用

- 缺点：

  不能显式控制某结点是否遍历

  不能返回值

```
virtual void enterCompUnit(sysyExtParser::CompUnitContext * /*ctx*/) override { }
virtual void exitCompUnit(sysyExtParser::CompUnitContext * /*ctx*/) override { }

virtual void enterConstDeclDecl(sysyExtParser::ConstDeclDeclContext * /*ctx*/) override { }
virtual void exitConstDeclDecl(sysyExtParser::ConstDeclDeclContext * /*ctx*/) override { }
```

ANTLR - Visitor模式

- 主动遍历

  ```
  $antlr4 -visitor example.g4 -no-listener
  ```

- 优点：

  可控制子节点遍历与否，来对特定节点进行遍历

  动作代码与文法产生式解耦，利于文法产生式的重用

  可以返回自定义值

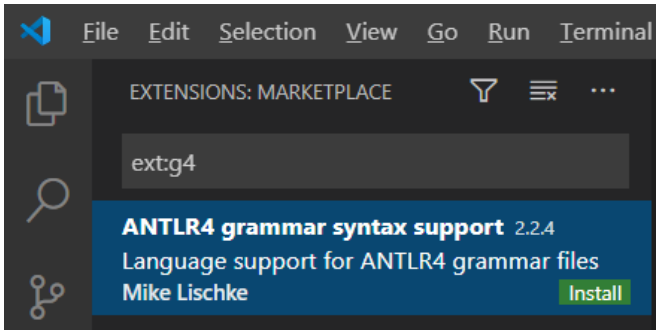
```
virtual antlrcpp::Any visitCompUnit(sysyExtParser::CompUnitContext *ctx) override {
  return visitChildren(ctx);
}

virtual antlrcpp::Any visitConstDeclDecl(sysyExtParser::ConstDeclDeclContext *ctx) override {
  return visitChildren(ctx);
}
```

Demo介绍

- Antlr：功能丰富

  有兴趣可以去官网进一步了解



- vscode插件
- demo compiler地址：

  http://124.16.71.65/compiler0/compiler.git

- CACT语言规范

  http://124.16.71.65/compiler0/material.git

Demo compiler

- dependencies/

  Antlr4-runtime等依赖

- grammar/

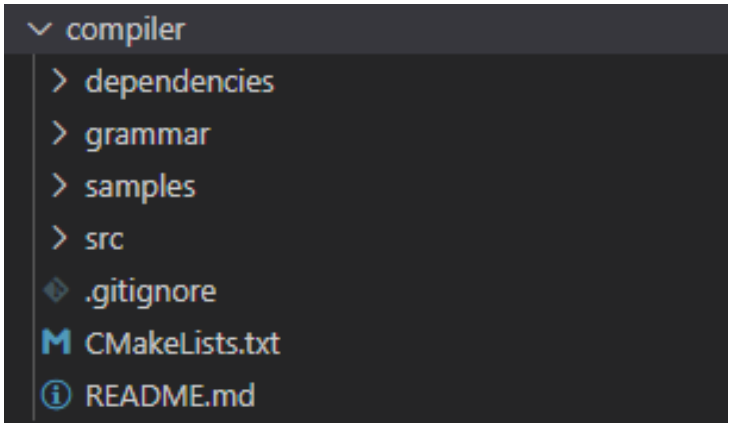
  .g4文件及生成的parser、lexer代码

- src/

  main.cpp、语义检查、中间代码生成、汇编代码生成

- samples/

- CMakeLists.txt



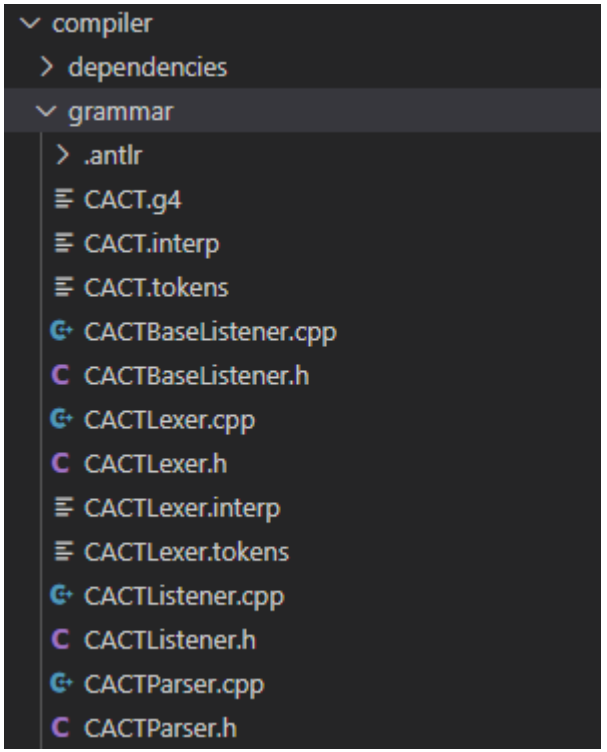
Demo compiler

- 生成lexer、parser
- 两种模式：

  listener

  visitor
- 编写.g4文件
- 执行antlr4 -Dlanguage=Cpp CACT.g4

  默认为listener模式



.g4文件——组成部分

- 可选项：

  options

  @header

- Parser
- Lexer
- Parser与Lexer没有显式的区分

```
1   grammar CACT;
2
3   // The language generated
4   options {
5       language = Cpp;
6   }
7
8   @header {
9       #include <vector>
10      #include <string>
11  }
12
13  /********** Parser **********/
```

CACT.g4 | CACTBaseListener.h ×

compiler > grammar > CACTBaseListener.h > ...

```
1
2       #include <vector>
3       #include <string>
4
5
6   // Generated from CACT.g4 by ANTLR 4.8
```

.g4文件——Lexer

- 定义终结符

  首字母大写

- 正则表达式
- fragment的使用
- skip

```
66  /********** Lexer **********/
67  BoolConst : 'true' | 'false';
68
69  Ident
70      : IdentNondigit [a-zA-Z_0-9]*
71      ;
72
73  fragment
74  IdentNondigit
75      : [a-zA-Z_]
76      ;
77
78  fragment
79  Digit
80      : [0-9]
81      ;
```

```
126 NewLine
127     : ('\r' '\n'? | '\n')
128       -> skip
129     ;
130
131 WhiteSpace
132     : [ \t]+
133       -> skip
134     ;
```

.g4文件——Parser

- 基本格式

  小写字母开头

  正则表达式

  EOF表示结束符

```
/********** Parser **********/
compUnit
    : (decl)+ EOF
    ;

decl
    : constDecl
    | varDecl
    ;

constDecl
    : 'const' bType constDef (',' constDef)* ';'
    ;

bType
    : 'int'
    | 'bool'
    ;
```

```
void SemanticAnalysis::exitConstDecl(CACTParser::ConstDeclContext * ctx)
{
    ctx->bType
    std::cout  bType                    BTypeContext * bTy…
```

.g4文件——Parser

- 可能会用到的特性

  子标签

```
51  constExp
52      locals[
53          int basic_or_array_and_type,
54      ]
55      : number            #constExpNumber
56      | BoolConst         #constExpBoolConst
57      ;
```

```
47  virtual void enterConstExpNumber(CACTParser::ConstExpNumberContext * /*ctx*/) override { }
48  virtual void exitConstExpNumber(CACTParser::ConstExpNumberContext * /*ctx*/) override { }
49
50  virtual void enterConstExpBoolConst(CACTParser::ConstExpBoolConstContext * /*ctx*/) override { }
51  virtual void exitConstExpBoolConst(CACTParser::ConstExpBoolConstContext * /*ctx*/) override { }
```

.g4文件——Parser

- 可能会用到的特性

  locals

  - 继承属性与综合属性

```
51  constExp
52      locals[
53          int basic_or_array_and_type,
54      ]
55      : number            #constExpNumber
56      | BoolConst         #constExpBoolConst
57      ;
58
59  number
60      locals[
61          int basic_or_array_and_type,
62      ]
63      : IntConst
64      ;
```

```
void SemanticAnalysis::exitConstExpNumber(CACTParser::ConstExpNumberContext * ctx)
{
    ctx->basic_or_array_and_type = ctx->number()->basic_or_array_and_type;
}
```

src文件

```
3   void SemanticAnalysis::enterConstDecl(CACTParser::ConstDeclContext * ctx)
4   {}
5   void SemanticAnalysis::exitConstDecl(CACTParser::ConstDeclContext * ctx)
6   {
7       std::cout << "const variable define: " << std::endl;
8       for(const auto & const_def : ctx->constDef())
9       {
10          std::cout << "\tname: " << const_def->Ident()->getText().c_str() \
11                    << " type: " << ctx->bType()->getText().c_str() << std::endl;
12      }
13  }
```

ANTLR

- PR001任务书.pdf
- PR001实验说明.pdf

总结

- 完成一个CACT到RV64GC的编译器
- 代码需要提交至课程指定的分组的gitlab上
- 语言规范和配套的支撑材料也会发布给同学们
- 课程网站的更新可能会有延迟，请大家关注gitlab

  语言规范和测试样例等支撑材料位于

  - http://124.16.71.65/compiler0/material

  Demo位于

  - http://124.16.71.65/compiler0/compiler