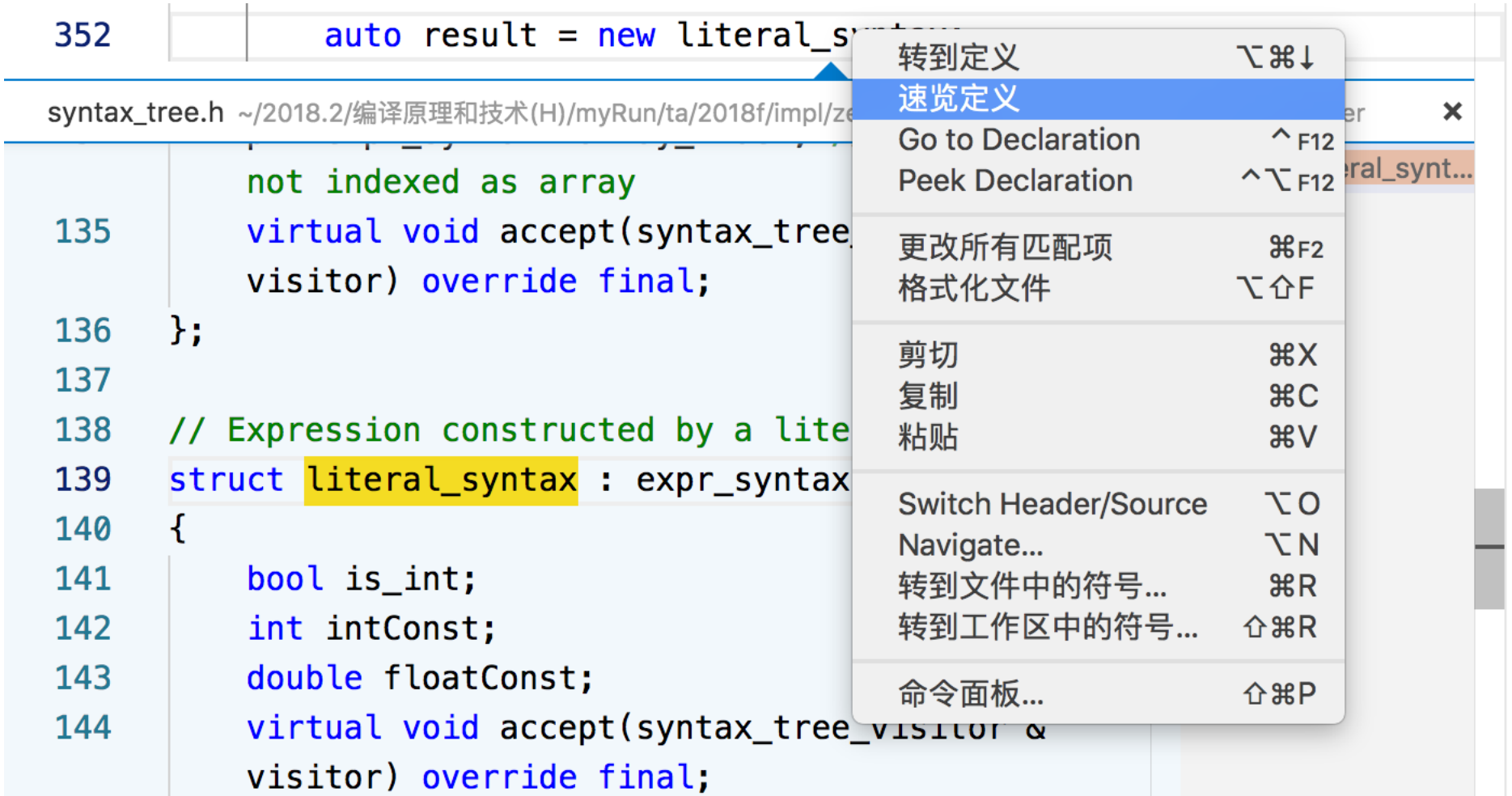
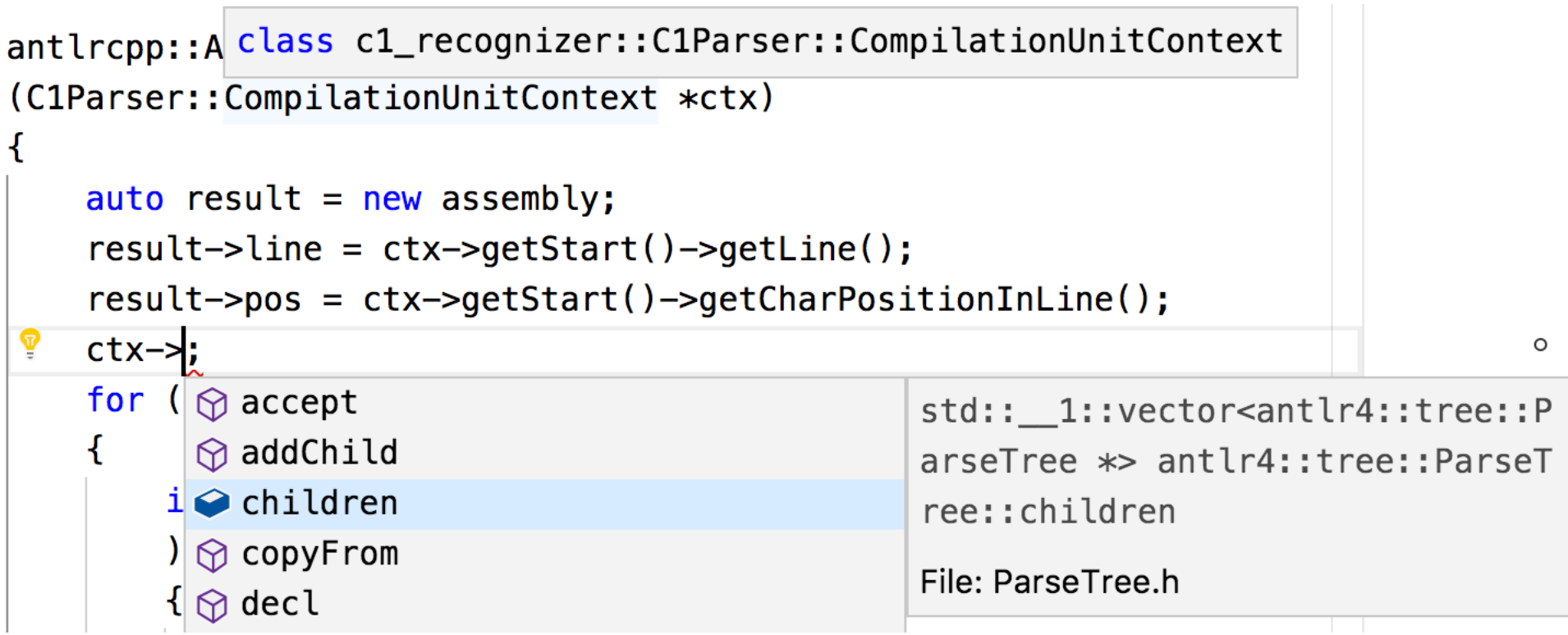
Lab1-3
Lab2-1

zevin

前言

吹爆 Intellisense


352
auto result = new literal_s
syntax_tree.h ~/2018.2/编译原理和技术(H)/myRun/ta/2018f/impl/ze
not indexed as array
135
virtual void accept(syntax_tree
visitor) override final;
136
};
137
138
// Expression constructed by a lite
139
struct literal_syntax : expr_syntax
140
{
141
bool is_int;
142
int intConst;
143
double floatConst;
144
virtual void accept(syntax_tree_visitor &
visitor) override final;
转到定义 ⌥⌘↓
速览定义
Go to Declaration ^F12
Peek Declaration ^⌥F12
更改所有匹配项 ⌘F2
格式化文件 ⌥⇧F
剪切 ⌘X
复制 ⌘C
粘贴 ⌘V
Switch Header/Source ⌥O
Navigate... ⌥N
转到文件中的符号... ⌘R
转到工作区中的符号... ⇧⌘R
命令面板... ⇧⌘P


class c1_recognizer::C1Parser::CompilationUnitContext
10 antlrcpp::A
(C1Parser::CompilationUnitContext *ctx)
11 {
12 auto result = new assembly;
13 result->line = ctx->getStart()->getLine();
14 result->pos = ctx->getStart()->getCharPositionInLine();
15 ctx->;
16 for (
17 {
18 i
)
19 {
accept
addChild
children
copyFrom
decl
std::__1::vector<antlr4::tree::ParseTree *> antlr4::tree::ParseTree::children
File: ParseTree.h

```
10  antlrcpp::Any syntax_tree_builder::visitCompilationUnit
    (C1Parser::CompilationUnitContext *ctx)
11  {
12      auto result = new assembly;
13      result->line = ctx->getStart()->getLine();
14      result->pos = ctx->getStart()->getCharPositionInLine();
15      ctx->;
16      for (
17      {
18          i
            )
19          {
20
21
22
23          }
```

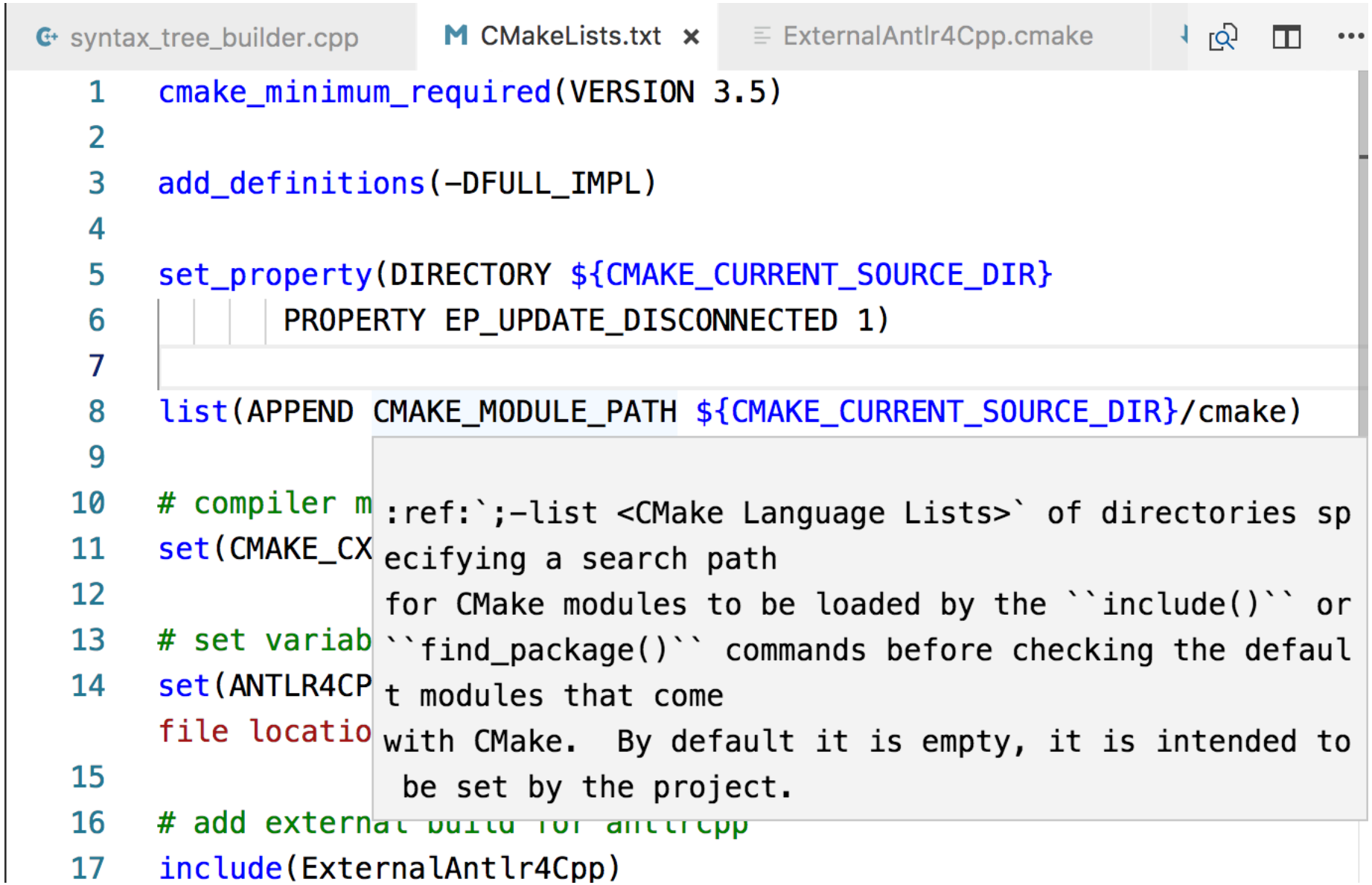
depth
EMPTY
enterRule
exception
exitRule
funcdef
getAltNumber
getRuleContext
getRuleContexts
getRuleIndex
getSourceInterval
getStart

```
size_t antlr4::RuleContext::getR
uleIndex() const

+1 overload

File: RuleContext.h
```

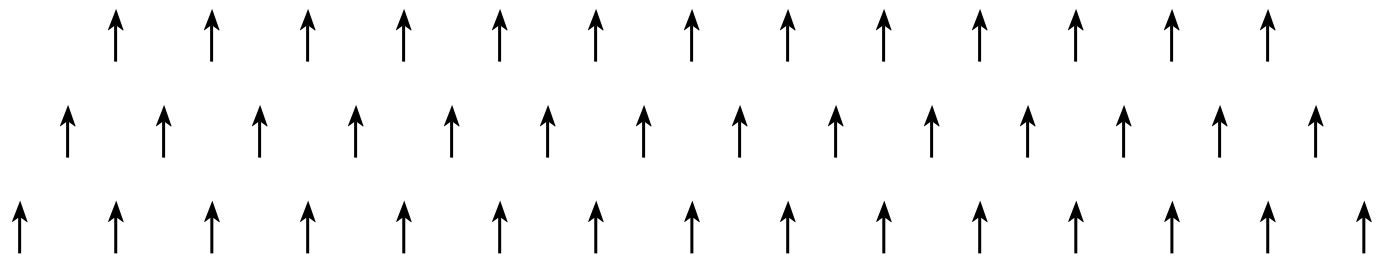
```
>();

obal_def_syntax>(def)
```

- 编译
  在命

```
# 在
```


syntax_tree_builder.cpp
CMakeLists.txt
ExternalAntlr4Cpp.cmake
1 cmake_minimum_required(VERSION 3.5)
2
3 add_definitions(-DFULL_IMPL)
4
5 set_property(DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
6 PROPERTY EP_UPDATE_DISCONNECTED 1)
7
8 list(APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake)
9
10 # compiler m
11 set(CMAKE_CX
12
13 # set variab
14 set(ANTLR4CP
file locatio
15
16 # add external build for antlrcpp
17 include(ExternalAntlr4Cpp)
:ref:`;-list <CMake Language Lists>` of directories specifying a search path
for CMake modules to be loaded by the ``include()`` or ``find_package()`` commands before checking the default modules that come
with CMake. By default it is empty, it is intended to be set by the project.

代码效率Max



Lab1-3

Lab1-3

- 分析项目结构
- 学习并理解visitor pattern
- 完成代码
- 回答问题

Lab1-3 分析项目结构 - *CMakeLists.txt*

Lab1-3 分析项目结构 - *CMakeLists.txt*

1. 设置版本,路径等参数:

如:

```
8   list(APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake)
```

```
11  set(CMAKE_CXX_STANDARD 14)
```

Lab xt

1. 设

如：

syntax_tree_builder.cpp | CMakeLists.txt | ExternalAntlr4Cpp.cmake

```
1  cmake_minimum_required(VERSION 3.5)
2
3  add_definitions(-DFULL_IMPL)
4
5  set_property(DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
6               PROPERTY EP_UPDATE_DISCONNECTED 1)
7
8  list(APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake)
9
10 # compiler m
11 set(CMAKE_CX
12
13 # set variab
14 set(ANTLR4CP
   file locatio
15
16 # add external build for antlrcpp
17 include(ExternalAntlr4Cpp)
```

```
:ref:`;-list <CMake Language Lists>` of directories sp
ecifying a search path
for CMake modules to be loaded by the ``include()`` or
``find_package()`` commands before checking the defaul
t modules that come
with CMake.  By default it is empty, it is intended to
 be set by the project.
```

Lab1-3 分析项目结构 - *CMakeLists.txt*

1. 设置版本,路径等参数
2. build antlr4cpp:

找到ExternalAntlr4Cpp.cmake, load and run.

```
16   # add external build for antlrcpp
17   include(ExternalAntlr4Cpp)
```

Lab1-3 分析项目结构 - *CMakeLists.txt*

1. 设置版本,路径等参数
2. build antlr4cpp
3. 用antlr4cpp处理lexer和grammar得到target的依赖

```
24  # Call macro to add lexer and grammar to your build dependencies.
25  antlr4cpp_process_grammar(c1_recognizer c1_recognizer
26    ${CMAKE_CURRENT_SOURCE_DIR}/grammar/C1Lexer.g4
27    ${CMAKE_CURRENT_SOURCE_DIR}/grammar/C1Parser.g4)
```

Lab

txt

1. 设置

2. bui

3. 用ar

```
168  ############ Generate runtime #################
169  # macro to add dependencies to target
170  #
171  # Param 1 project name
172  # Param 1 namespace (postfix for dependencies)
173  # Param 2 Lexer file (full path)
174  # Param 3 Parser File (full path)
175  #
176  # output
177  #
178  # antlr4cpp_src_files_{namespace} - src files for add_executable
179  # antlr4cpp_include_dirs_{namespace} - include dir for generated
     headers
180  # antlr4cpp_generation_{namespace} - for add_dependencies tracking
181
182  macro(antlr4cpp_process_grammar
183      antlr4cpp_project
184      antlr4cpp_project_namespace
185      antlr4cpp_grammar_lexer
186      antlr4cpp_grammar_parser)
187
```

Lab1-3 分析项目结构 - *CMakeLists.txt*

1. 设置版本,路径等参数
2. build antlr4cpp
3. 用antlr4cpp处理lexer和grammar得到target的依赖
4. 添加rapidjson的引用

Lab1-3 分析项目结构 - *CMakeLists.txt*

1. 设置版本,路径等参数
2. build antlr4cpp
3. 用antlr4cpp处理lexer和grammar得到target的依赖
4. 添加rapidjson的引用
5. 编译生成的grammar文件,添加依赖关系,链接

Lab1-3 分析项目结构 - *CMakeLists.txt*

1. 设置版本,路径等参数
2. build antlr4cpp
3. 用antlr4cpp处理lexer和grammar得到target的依赖
4. 添加rapidjson的引用
5. 编译生成的grammar文件,添加依赖关系,链接
6. 编译生成可执行文件clr_test

Lab1-3 分析项目结构 - *CMakeLists.txt*

1. 设置版本,路径等参数
2. build antlr4cpp
3. 用antlr4cpp处理lexer和grammar得到target的依赖
4. 添加rapidjson的引用
5. 编译生成的grammar文件,添加依赖关系,链接
6. 编译生成可执行文件clr_test
7. 生成clrecognizer的installation rules,以支持Lab2.

Lab1-3 分析项目结构 - *功能模块*

Lab1-3 分析项目结构 - *功能模块*

1.recognizer: 主要模块

- 通过 C1Lexer lexer(input) 以及CommonTokenStream tokens(&lexer)得到 token 流;
- 通过 C1Parser parser(&tokens) 以及 parser.compilationUnit() 得到 parse tree;
- 通过 ast = ast_builder(tree) 调用 syntax_tree_builder，构建 AST;
- 得到的 AST 在main.cpp 转换为 JSON 格式输出;

Lab1-3 分析项目结构 - *功能模块*

1.recognizer: 主要模块

2.antlr4cpp: .g4 -> c++源代码

通过.g4生成 lexer 和 parser 的 C++ 代码

Lab1-3 分析项目结构 - *功能模块*

1. recognizer: 主要模块
2. antlr4cpp: .g4 -> c++源代码
3. syntax_tree_builder: parse tree -> AST

本次实验主要需要完成的部分,通过访问者模式遍历parse tree,生成AST

Lab1-3 分析项目结构 - *功能模块*

1. recognizer: 主要模块
2. antlr4cpp: .g4 -> c++源代码
3. syntax_tree_builder: parse tree -> AST
4. syntax_tree_serializer: AST -> .json

将 Recognizer 中构建好的 AST 进行序列化，通过 rapidjson 得到 JSON 格式的输出

Labl-3 visitor pattern

Labl-3 visitor pattern

Lab2-2 也会涉及到

Labl-3 visitor pattern

- visitor pattern 解决的问题?
  - It should be possible to define a new operation for (some) classes of an object structure without changing the classes.
  - 每次都弄一个新的subclass不够灵活
- 怎么做的?

Lab1-3 结合Lab1-3理解Visitor Pattern

在 visitExp() 中,当表达式是 number 时, 调用 visit(number)

```
// syntax_tree_builder.cpp
antlrcpp::Any syntax_tree_builder::visitExp(C1Parser::ExpContext *ctx)
{
  ...
    if (auto number = ctx->number())
        return visit(number); // <-这里
  ...
}
```

Lab1-3 结合Lab1-3理解Visitor Pattern

在 visitExp() 中,当表达式是 number 时, 调用 visit(number)

syntax_tree_builder 继承自AbstractParseTreeVisitor

```
13  class syntax_tree_builder : public C1ParserBaseVisitor
```

```
17  class  C1ParserBaseVisitor : public C1ParserVisitor {
```

```
17  class  C1ParserVisitor : public
    antlr4::tree::AbstractParseTreeVisitor {
```

```
13  class ANTLR4CPP_PUBLIC AbstractParseTreeVisitor : public
    ParseTreeVisitor {
14  public:
15    /// The default implementation calls <seealso
      cref="ParseTree#accept"/> on the
16    /// specified tree.
17    virtual antlrcpp::Any visit(ParseTree *tree) override {
18      return tree->accept(this);
19    }
```

Lab1-3 结合Lab1-3理解Visitor Pattern

在 visitExp() 中,当表达式是 number 时, 调用 visit(number)
syntax_tree_builder 继承自AbstractParseTreeVisitor
所以继续执行number->accept(syntax_tree_builder*)

```
13  class ANTLR4CPP_PUBLIC AbstractParseTreeVisitor : public
    ParseTreeVisitor {
14  public:
15    /// The default implementation calls <seealso
      cref="ParseTree#accept"/> on the
16    /// specified tree.
17    virtual antlrcpp::Any visit(ParseTree *tree) override {
18      return tree->accept(this);
19    }
```

Lab1-3 结合Lab1-3理解Visitor Pattern

在 visitExp() 中,当表达式是 number 时, 调用 visit(number)
syntax_tree_builder 继承自AbstractParseTreeVisitor
所以继续执行number->accept(syntax_tree_builder*)
继续执行syntax_tree_builder->visitNumber(number)

```
1645  antlrcpp::Any C1Parser::NumberContext::accept
      (tree::ParseTreeVisitor *visitor) {
1646    if (auto parserVisitor = dynamic_cast<C1ParserVisitor*>
        (visitor))
1647      return parserVisitor->visitNumber(this);
1648    else
1649      return visitor->visitChildren(this);
1650  }
```

Lab1-3 结合Lab1-3理解Visitor Pattern

在 visitExp() 中,当表达式是 number 时, 调用 visit(number)
syntax_tree_builder 继承自AbstractParseTreeVisitor
所以继续执行number->accept(syntax_tree_builder*)
继续执行syntax_tree_builder->visitNumber(number)

好处?

Lab1-3 完成代码

Lab1-3 完成代码

- 要得到TerminalNode
  - 直接ctx-> RightBracket()
    - // not exists -> nullptr
- 要得到NonTerminalNode
  - 直接ctx-> exp()
    - // not exists -> nullptr

Lab1-3 完成代码

- 判断具体是哪个产生式(alternative):
  - 可以用该alternative特有的terminalNode或者nonTerminalNode判断
  - terminalNode 举例:
    - 比如 ctx-> RightBracket()
    - // not exists -> nullptr -> bool : false
    - // exists -> not nullptr -> bool : true
  - nonTerminalNode 举例:
    - 比如 (decl | funcdef)+
    - 可以ctx->children得到vector
    - 这里vector的元素顺序不能打乱,否则影响程序语义.
    - 然后对其中元素,通过 dynamic_cast<ClParser::DeclContext *>()判断

Lab1-3 完成代码

- 1. block_syntax里只有stmt_syntax?decl呢?
- 2. var_def_stmt_syntax继承自stmt_syntax作为stmt的一种，但是EBNF文法描述中，VarDef并不是Stmt多个选择之一?

```
153   // Variable definition. Multiple of this would be both a
      statement and a global definition; however, itself only
154   // represents a single variable definition.
155   struct var_def_stmt_syntax : stmt_syntax, global_def_syntax
156   {
```

- syntax_tree.h里的var_def_stmt_syntax是具体实现,和理论文法不用一一对应.
- 可以体会一下这么实现带来的好处

Lab1-3 完成代码

- 类型问题 ：
  - std::bad_cast的错误指什么?antlrcpp::Any又是干嘛的?

> 这个问题可以从`syntax_tree_builder.cpp`的注释中得到解答.

```
// Returns antlrcpp::Any, which is constructable from any type.
// However, you should be sure you use the same type for packing and depacking the `Any` object.
// Or a std::bad_cast exception will rise.
// This function always returns an `Any` object containing a `expr_syntax *`.
antlrcpp::Any syntax_tree_builder::visitExp(C1Parser::ExpContext *ctx){
    ...
}
```

Lab1-3 完成代码

- 类型问题 ：
  - ptr_list应该怎么赋值？
  - 用vector的方法push_back(visit(somePointer).as<*_syntax *>())报错说没有匹配的函数。
- 没有匹配的函数是因为函数的参数类型不对
- 查看ptr_list定义

```
// Use unique postd::stringtype to reference stored objects
template <typename T>
using ptr = std::shared_ptr<T>;

// List of reference of type
template <typename T>
using ptr_list = std::vector<ptr<T>>;
```

Lab1-3 完成代码

- 类型问题 ：
  - ptr_list应该怎么赋值？
  - 用vector的方法push_back(visit(somePointer).as<*_syntax *>())报错说没有匹配的函数。
- 然后根据vector的类型,以及somePointer的类型,
- 如果vector是ptr_list<T>
- 就在 visit(somePointer).as<*_syntax *>() 外面再套一层类型转换
- 如果本身是智能指针,那就套dynamic_pointer_cast()或者static_pointer_cast().
- 如果是普通指针,那就套ptr()

Lab1-3 完成代码

- 类型问题 :
- 总结其实就是:
  - 调用的时候参数类型和函数声明的参数类型一致
  - 使用返回值的时候和函数定义最后return的类型一致

Lab1-3回答问题 - 三个函数

Lab1-3 回答问题 - 三个函数

- enterRecursionRule :作用是做**递归前**的准备工作
- unrollRecursionContexts :作用是做**递归结束后**的恢复和保存工作
- adaptivePredict :论文中描述的 adaptivePredict 过程。
- 对于产生式存在谓词的非终结符，利用上下文和栈进行预测，即 LL 分析；否则进行 SLL 分析（不需要用到栈）。

Lab1-3 回答问题 - 错误处理

Lab1-3 回答问题 - 错误处理

具体实现:

DefaultErrorStrategy

代码实现及详尽解释见DefaultErrorStrategy.(cpp/h)

```
11  namespace antlr4 {
12
13    /**
14     * This is the default implementation of {@link
      ANTLRErrorStrategy} used for
15     * error reporting and recovery in ANTLR parsers.
16     */
17    class ANTLR4CPP_PUBLIC DefaultErrorStrategy : public
      ANTLRErrorStrategy {
```

Lab1-3 回答问题 - 错误处理

具体实现:

DefaultErrorStrategy.

代码实现及详尽解释见DefaultErrorStrategy.(cpp/h)

接口:

DefaultErrorStrategy::reportError()

DefaultErrorStrategy::recover()

DefaultErrorStrategy::consumeUntil()

等

Lab1-3 回答问题 - 三个*LL

- SLL：

The strong $LL(k)$ grammars are a subset of the $LL(k)$ grammars that can be parsed without knowledge of the left-context of the parse. That is, each parsing decision is based only on the next k tokens of the input for the current nonterminal that is being expanded. A definition of the strong $LL(k)$ grammars follows.

Definition: A grammar $G = (N, T, P, S)$ is said to be strong $LL(k)$ for some fixed natural number k if for all nonterminals A, and for any two distinct A-productions in the grammar

A --> α

A --> β

$$FIRST_k(\alpha FOLLOW_k(A)) \cap FIRST_k(\beta FOLLOW_k(A)) = \emptyset$$

Lab1-3 回答问题 - 三个*LL

- LL：

> An LL parser is called an $LL(k)$ parser if it uses k tokens of lookahead when parsing a sentence. If such a parser exists for a certain grammar and it can parse sentences of this grammar without backtracking then it is called an $LL(k)$ grammar.
>
> ...
>
> Let G be a context-free grammar and $k \geq 1$. We say that G is $LL(k)$, if and only if for any two leftmost derivations:
>
> 1. $S \Rightarrow ... \Rightarrow \omega A \alpha \Rightarrow ... \Rightarrow \omega \beta \alpha \Rightarrow ... \Rightarrow \omega x$
> 2. $S \Rightarrow ... \Rightarrow \omega A \alpha \Rightarrow ... \Rightarrow \omega \gamma \alpha \Rightarrow ... \Rightarrow \omega y$
>
> Following conditions holds: Prefix of the string x of length k equals the prefix of the y of length k implies $\beta = \gamma$

Lab1-3 回答问题 - 三个*LL

- $LL(k)$是不需要回溯的，其根据已经使用的产生式以及读入的k个符号来判断使用哪一个产生式。
- 而$SLL(k)$不需要其上文，只需要根据接下来的k个符号即可判断使用哪一个产生式，$SLL(k)$是L$LL(k)$的一个子集，从而$SLL(k)$也不需要回溯。
- 综上
- SLL 和 LL 是不试探回溯的
- 并且，$SLL(k)$、$LL(k)$均不允许二义文法

Lab1-3 回答问题 - 三个*LL

- ALL(*)采用的方法是在每个不确定的决策点启动多个子分析器，每个分析器对应一种可能。
- 它们以伪并行的方式遍历所有的可能，当某个子分析器匹配失败的时候即被杀掉，直到最后只剩一个匹配成功的子分析器。
- 如果直到文件结尾还有多个存活的子分析器，则分析器会优先选择使用了编号较小的产生式的那个分析器。
- 从而，ALL(*)没有回溯，但是允许文法二义。
- *不是指两阶段ALL(*)*

Lab2-1 预热

- 准备:将llvm-install/bin添加到PATH.
  - *注意环境变量添加记得":"*
    比如export LD_LIBRARY_PATH=./Libs_for_clr_ref

- 准备:将llvm-install/bin添加到PATH.
- 观察clang -S -emit-llvm输出的结果
  - 可以发现:
    - 每个函数对应一个独立的计数过程.
    - 每个label占用一个计数值.
    - 存储return值占用一个技术值.
    - 每个BB对应没有分支的顺序的过程.
    - 每个BB除了最后一个,都以br结束.
    - 等

- 准备:将llvm-install/bin添加到PATH.
- 观察clang -S -emit-llvm输出的结果
- 查看http://llvm.org/docs/LangRef.html
  - 理解alloca,align
  - 理解add,nsw,store,load,call
  - 理解if,br,ret
  - 理解类型,函数类型,函数定义

- 准备:将llvm-install/bin添加到PATH.
- 观察clang -S -emit-llvm输出的结果
- 查看http://llvm.org/docs/LangRef.html
- 阅读Kaleidoscope 的例子
  - (http://llvm.org/docs/tutorial/LangImpl03.html)
  - 阅读Code Generation Setup
  - 阅读Function Code Generation
  - 理解函数定义方式,参数使用方式
  - 理解BB的基本流程
    - BasicBlock::Create
    - 先SetInsertPoint
    - 后CreateBr

- 准备:将llvm-install/bin添加到PATH.
- 观察clang -S -emit-llvm输出的结果
- 查看http://llvm.org/docs/LangRef.html
- 阅读Kaleidoscope 的例子
- 合理配置Intellisense,参考文档获取所需API

再次
吹爆 Intellisense

Thanks

最后!
代码注释!!!

```
102     // for cond
103     [illegible]
104     [illegible]ip, 4);
105     [illegible]0));
106     [illegible]
107
108     // for loop
109     [illegible]
110     auto x = [illegible]xp, 4);
111     [illegible]4);
112     [illegible]{ i1 });
113     [illegible]
114     [illegible]
115
116     // i++
117     [illegible]
118     auto i2 [illegible]));
119     [illegible].CreateAlignedStore(i3, ip, 4);
120     [illegible]);
121
122     // ret
123     [illegible]
124     [illegible]);
125     [illegible]
126
127     theModule->print(outs(), nullptr);
128     delete theModule;
129     return 0;
130 }
```

```
1
2  define i32 @fib(i32 %n) {
3      %1 = alloca i32, align 4          ; %1 for return va
4      %2 = alloca i32, align 4          ; %2 store pa
5      store i32 %0, i32* %2, align 4    ; put n t
6      %3 = load i32, i32* %2, align 4   ; load %n
7      %4 = icmp eq i32 %3, 0            ; %4 is
8      br i1 %4, label %5, label %6      ; if %4, jump to %5, else jump to %6
9
10 ; <label>:5                           ; repre      0. return 0
11     store i32 0, i32* %1, align 4     ; put co      return value's pla
12     br label %16                      ; jump t
13
14 ; <label>:6                           ; repres      = 0. or 1
15     %7 = icmp eq i32 %3, 1            ; %7 is
16     br i1 %7, label %8, label %9      ; if %7,      to
17
18 ; <label>:8                           ; repres      == 1. return 1
19     store i32 1, i32* %1, align 4     ; put co      into return value's pla
20     br label %16                      ; jump t
21
22 ; <label>:9                           ; repres      != 1. return fib(n - 1) + fib(n - 2)
23     %10 = sub nsw i32 %3, 1           ; %10 is
24     %11 = call i32 @fib(i32 %10)      ; %11 is fib(
25     %12 = sub nsw i32 %3, 2           ; %12 is
26     %13 = call i32 @fib(i32 %12)      ; %13 is fib(n -
27     %14 = add nsw i32 %11, %13        ; %14 is fib(n - 1) + fib(n -
28     store i32 %14, i32* %1, align 4   ; put %14
29     br label %16                      ; jump to
30 ; <label>:16
31     %17 = load i32, i32* %1, align 4  ; get
32     ret i32 %17                       ; return
33 }
34
35 define i32 @main() {
```