中国科学技术大学
University of Science and Technology of China

使用 ANTLR 为 C1 构造生成AST的解析器

Lab：ParseTree =>AST

□ 使用 ANTLR 为 C1 语言构造生成AST的解析器

1. 编写C1语言的词法描述文件，参见 c1recognizer/grammar/C1Lexer.g4

```
lexer grammar C1Lexer;

tokens {
    Comma,
    SemiColon,
    Assign,
    LeftBracket,
    RightBracket,
    ……
    Identifier,
    FloatConst,
    IntConst
}
```

```
Comma: ',';
SemiColon: ';';
Assign: '=';
LeftBracket: '[';
RightBracket: ']';
……
Identifier: [_a-zA-Z] [_0-
9a-zA-Z]*;
……

LineComment: ('//' | '/\\'
('\r'? '\n') '/') ~[\r\n\\]*
('\\' ('\r'? '\n')?
~[\r\n\\]*)* '\r'? '\n' ->
```

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

WhiteSpace: [ \t\r\n]+ ->
skip;
```

LineComment、BlockComment、WhiteSpace不属于Tokens，即词法分析器识别但不返回记号给语法分析器。

https://www.educoder.net/shixuns/qix6mfn3/challenges

Lab：ParseTree =>AST

□ 使用 ANTLR 为 C1 语言构造生成AST的解析器

2. 编写C1语言的语法描述文件，参见 c1recognizer/grammar/C1Parser.g4

```
parser grammar C1Parser;
options { tokenVocab = C1Lexer; }

compilationUnit: ;
decl: ;
constdecl: ;
constdef: ;
vardecl: ;
vardef: ;
funcdef: ;
block: ;
stmt: ;
lval: ;
cond: ;
```

```
exp:
    (Plus | Minus) exp
    | exp (Multiply | Divide | Modulo) exp
    | exp (Plus | Minus) exp
    | LeftParen exp RightParen
    | number
;
number: ;
```

- 需要自行补充文法规则
- 推荐在描述语法规则时不加语义动作代码
- 分析器将生成解析树ParseTree

https://www.educoder.net/shixuns/qix6mfn3/challenges

Lab：ParseTree =>AST

需要根据解析树ParseTree构建语法树，即实现syntax_tree_builder

□ 使用 ANTLR 为 C1 语言构造解析器

3. C1 语言识别器源码，参见c1recognizer/src/recognizer.cpp

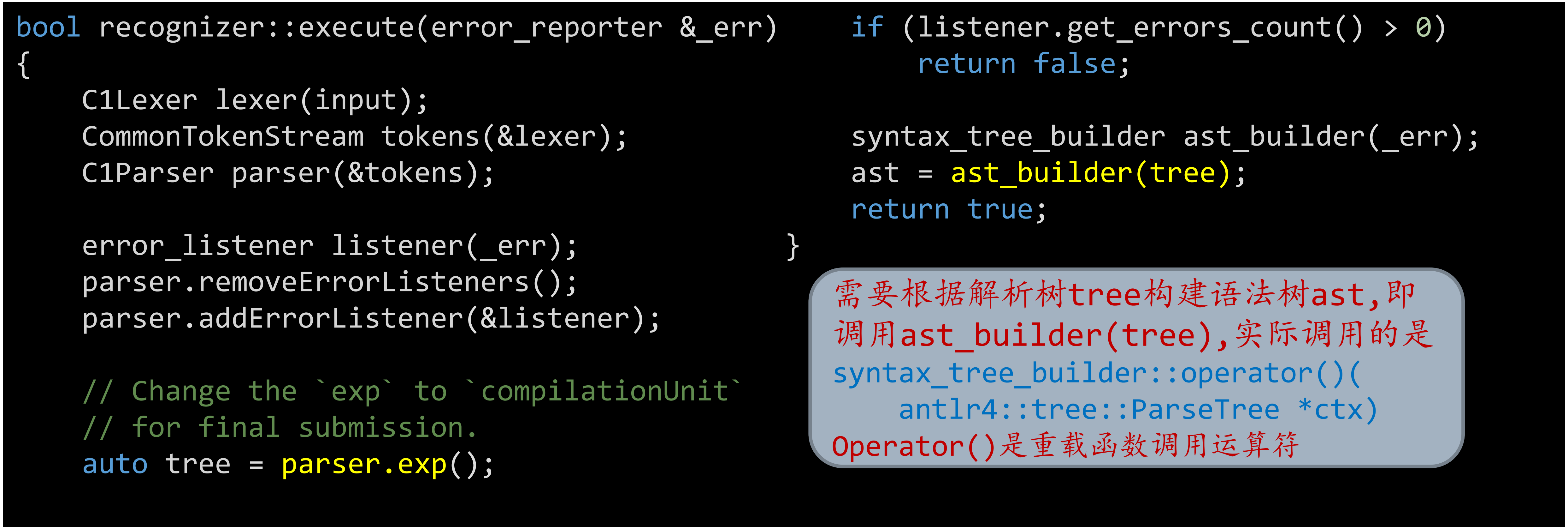
```cpp
#include <antlr4-runtime.h>
#include <C1Lexer.h>
#include <C1Parser.h>
#include <c1recognizer/recognizer.h>

#include <c1recognizer/syntax_tree_builder.h>
#include <c1recognizer/error_listener.h>

using namespace c1_recognizer;
using namespace syntax_tree;

using namespace antlr4;
using namespace antlrcpp;

recognizer::recognizer(const std::string
       &input_string) : ast(nullptr) {
    input = new ANTLRInputStream(input_string);
}

recognizer::recognizer(std::istream
                 &input_stream) : ast(nullptr)
{
    input = new ANTLRInputStream(input_stream);
}

std::shared_ptr<syntax_tree::syntax_tree_node>
recognizer::get_syntax_tree() { return ast; }

recognizer::~recognizer()
{
    delete input;
}
```

https://www.educoder.net/shixuns/qix6mfn3/challenges

Lab：ParseTree =>AST

□ 使用 ANTLR 为 C1 语言构造生成AST的解析器

3. C1 语言识别器源码，参见c1recognizer/src/recognizer.cpp



```cpp
bool recognizer::execute(error_reporter &_err)
{
    C1Lexer lexer(input);
    CommonTokenStream tokens(&lexer);
    C1Parser parser(&tokens);

    error_listener listener(_err);
    parser.removeErrorListeners();
    parser.addErrorListener(&listener);

    // Change the `exp` to `compilationUnit`
    // for final submission.
    auto tree = parser.exp();
    if (listener.get_errors_count() > 0)
        return false;

    syntax_tree_builder ast_builder(_err);
    ast = ast_builder(tree);
    return true;
}
```

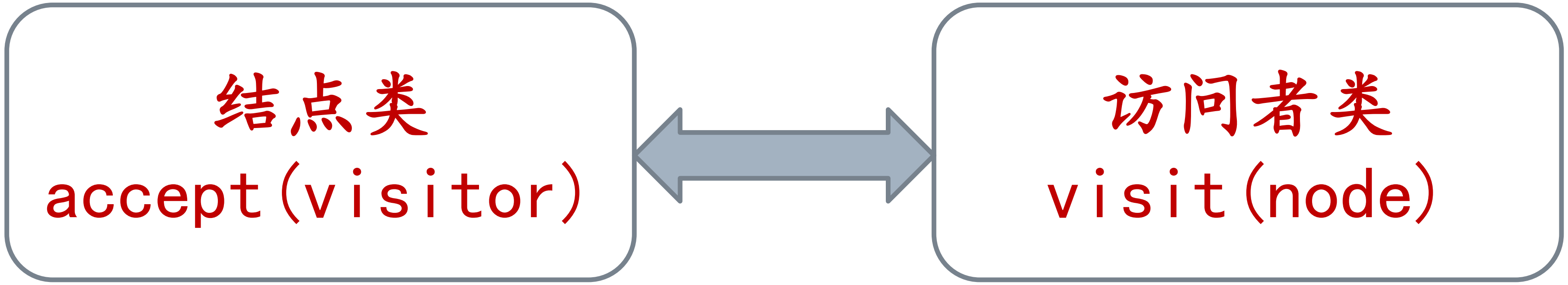
需要根据解析树tree构建语法树ast，即调用ast_builder(tree)，实际调用的是
syntax_tree_builder::operator()(
    antlr4::tree::ParseTree *ctx)
Operator()是重载函数调用运算符

https://www.educoder.net/shixuns/qix6mfn3/challenges

Lab：ParseTree =>AST

□ 访问者模式

- 解耦结点的结构与访问的方式
- 方便扩展结点类
- 方便增加更多的访问者类



结点类
accept(visitor)

访问者类
visit(node)

Lab：ParseTree =>AST

- □ **解析树ParseTree的定义：属于antlr4-runtime**(antlr4-runtime/tree/ParseTree.h)

```
class ANTLR4CPP_PUBLIC ParseTree {
  public:
    ParseTree();
    ……
    ParseTree *parent;
    std::vector<ParseTree *> children;
    virtual antlrcpp::Any accept(ParseTreeVisitor *visitor) = 0;
}
```

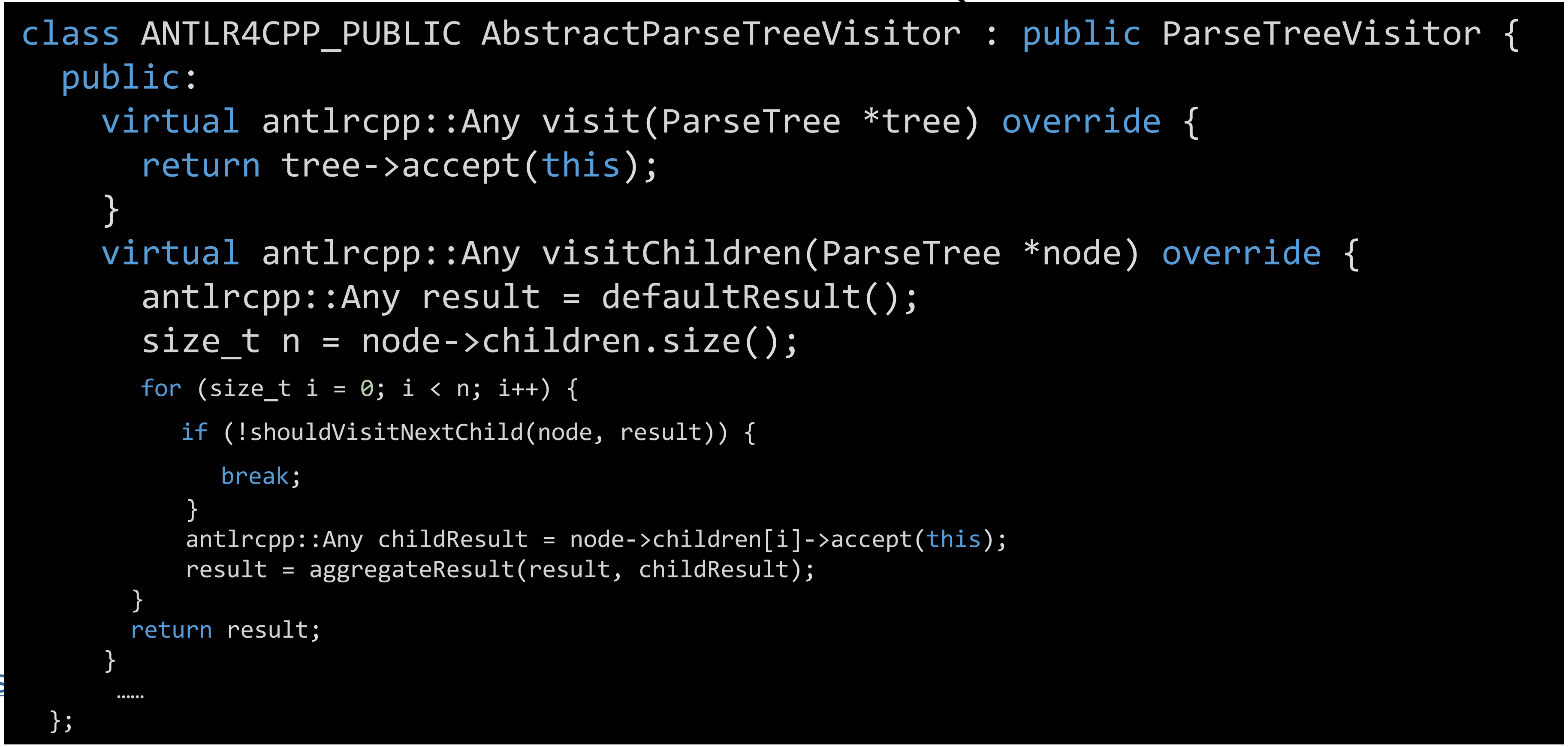
- □ **ParseTreeVisitor的定义：**(antlr4-runtime/tree/ParseTreeVisitor.h)

```
 class ANTLR4CPP_PUBLIC ParseTreeVisitor {
  public:
    virtual ~ParseTreeVisitor();
    virtual antlrcpp::Any visit(ParseTree *tree) = 0;
    virtual antlrcpp::Any visitChildren(ParseTree *node) = 0;
    virtual antlrcpp::Any visitTerminal(TerminalNode *node) = 0;
    virtual antlrcpp::Any visitErrorNode(ErrorNode *node) = 0;
  };
```

https://www.educoder.net/shixuns/qix6mfn3/challenges

Lab：ParseTree =>AST

- □ **解析树ParseTree的定义：属于antlr4-runtime(**antlr4-runtime/tree/ParseTree.h**)**
- □ **ParseTreeVisitor的定义：(**antlr4-runtime/tree/ParseTreeVisitor.h**)**
- □ **AbstracParseTreeVisitor的定义：(**antlr4-runtime/tree/AbstractParseTreeVisitor.h**)**



```
class ANTLR4CPP_PUBLIC AbstractParseTreeVisitor : public ParseTreeVisitor {
  public:
    virtual antlrcpp::Any visit(ParseTree *tree) override {
      return tree->accept(this);
    }
    virtual antlrcpp::Any visitChildren(ParseTree *node) override {
      antlrcpp::Any result = defaultResult();
      size_t n = node->children.size();
      for (size_t i = 0; i < n; i++) {
        if (!shouldVisitNextChild(node, result)) {
          break;
        }
        antlrcpp::Any childResult = node->children[i]->accept(this);
        result = aggregateResult(result, childResult);
      }
      return result;
    }
     ……
  };
```

https

Lab：ParseTree =>AST

- 根据文法生成的解析器，位于src/antlr4cpp_generated_src/C1Parser下
  - **class C1Parser : public antlr4::Parser**

包含class CompilationUnitContext : public antlr4::ParserRuleContext 等Context类

```
class  CompilationUnitContext : public antlr4::ParserRuleContext {
 public:
   CompilationUnitContext(antlr4::ParserRuleContext *parent, size_t invokingState);
   virtual size_t getRuleIndex() const override;
   antlr4::tree::TerminalNode *EOF();
   std::vector<DeclContext *> decl();
   DeclContext* decl(size_t i);
   std::vector<FuncdefContext *> funcdef();
   FuncdefContext* funcdef(size_t i);

   virtual void enterRule(antlr4::tree::ParseTreeListener *listener) override;
   virtual void exitRule(antlr4::tree::ParseTreeListener *listener) override;

   virtual antlrcpp::Any accept(antlr4::tree::ParseTreeVisitor *visitor) override;

 };
```

https://ww

Lab：ParseTree =>AST

- 根据文法生成的解析器，位于src/antlr4cpp_generated_src/C1Parser下
  - **class C1ParserVisitor : public antlr4::tree::AbstractParseTreeVisitor**

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

  /**
   * Visit parse trees produced by C1Parser.
   */
    virtual antlrcpp::Any visitCompilationUnit(C1Parser::CompilationUnitContext *context) = 0;

    virtual antlrcpp::Any visitDecl(C1Parser::DeclContext *context) = 0;


    ……

}
```

https://www.educoder.net/shixuns/qix6mfn3/challenges

Lab：ParseTree =>AST

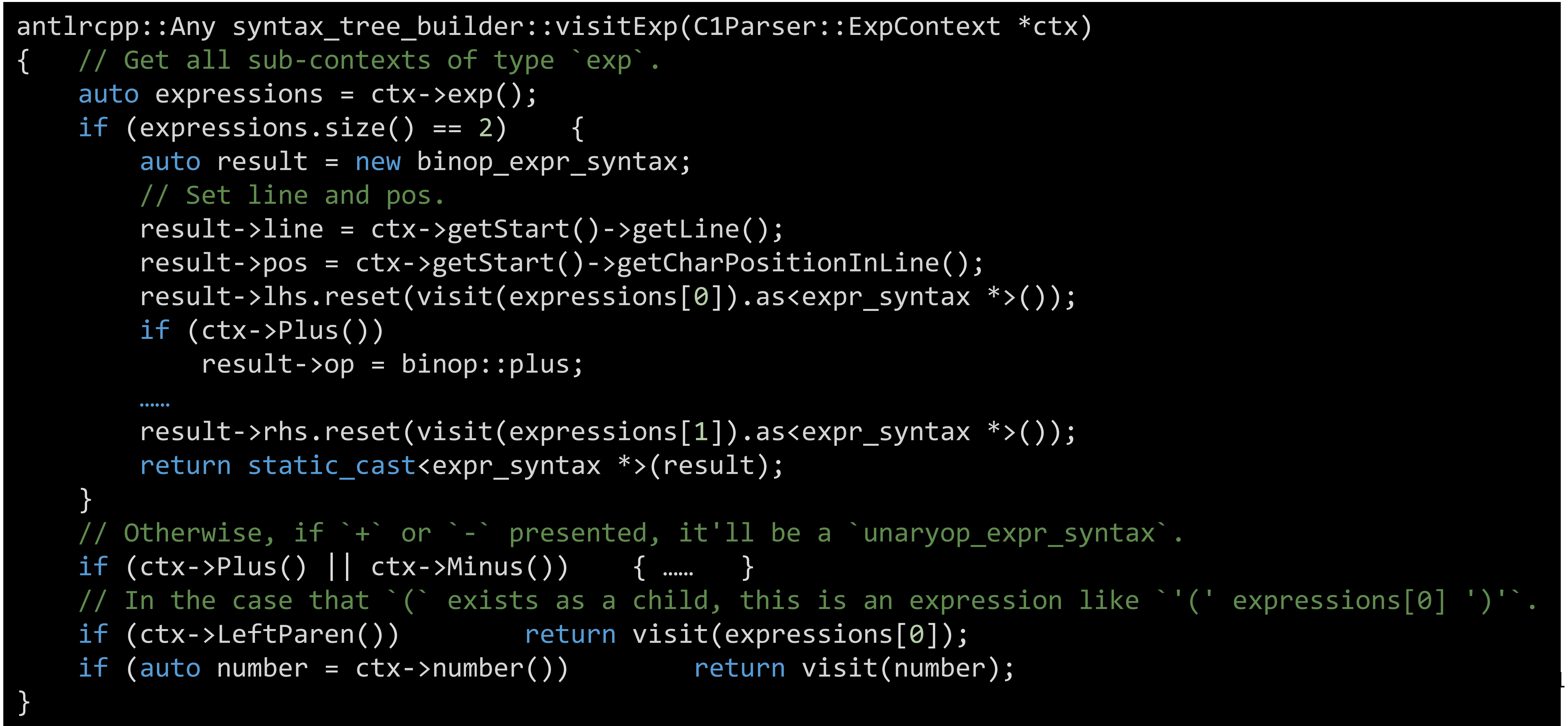
□ 由解析树构造AST

- c1recognizer/include/c1recognizer/syntax_tree_builder.h

```
namespace c1_recognizer {
namespace syntax_tree {
class syntax_tree_builder : public C1ParserBaseVisitor {
  public:
    syntax_tree_builder(error_reporter &_err);

    virtual antlrcpp::Any visitCompilationUnit(C1Parser::CompilationUnitContext *ctx) override;
    virtual antlrcpp::Any visitDecl(C1Parser::DeclContext *ctx) override;
     ……
    virtual antlrcpp::Any visitExp(C1Parser::ExpContext *ctx) override;
    virtual antlrcpp::Any visitNumber(C1Parser::NumberContext *ctx) override;

    ptr<syntax_tree_node> operator()(antlr4::tree::ParseTree *ctx);

  private:
    error_reporter &err;
};
}
```

Lab：ParseTree =>AST

□ 由解析树构造ASTc1recognizer/src/syntax_tree_builder.cpp



```cpp
antlrcpp::Any syntax_tree_builder::visitExp(C1Parser::ExpContext *ctx)
{   // Get all sub-contexts of type `exp`.
    auto expressions = ctx->exp();
    if (expressions.size() == 2)    {
        auto result = new binop_expr_syntax;
        // Set line and pos.
        result->line = ctx->getStart()->getLine();
        result->pos = ctx->getStart()->getCharPositionInLine();
        result->lhs.reset(visit(expressions[0]).as<expr_syntax *>());
        if (ctx->Plus())
            result->op = binop::plus;
        ……
        result->rhs.reset(visit(expressions[1]).as<expr_syntax *>());
        return static_cast<expr_syntax *>(result);
    }
    // Otherwise, if `+` or `-` presented, it'll be a `unaryop_expr_syntax`.
    if (ctx->Plus() || ctx->Minus())    { ……   }
    // In the case that `(` exists as a child, this is an expression like `'(' expressions[0] ')'`.
    if (ctx->LeftParen())        return visit(expressions[0]);
    if (auto number = ctx->number())        return visit(number);
}
```

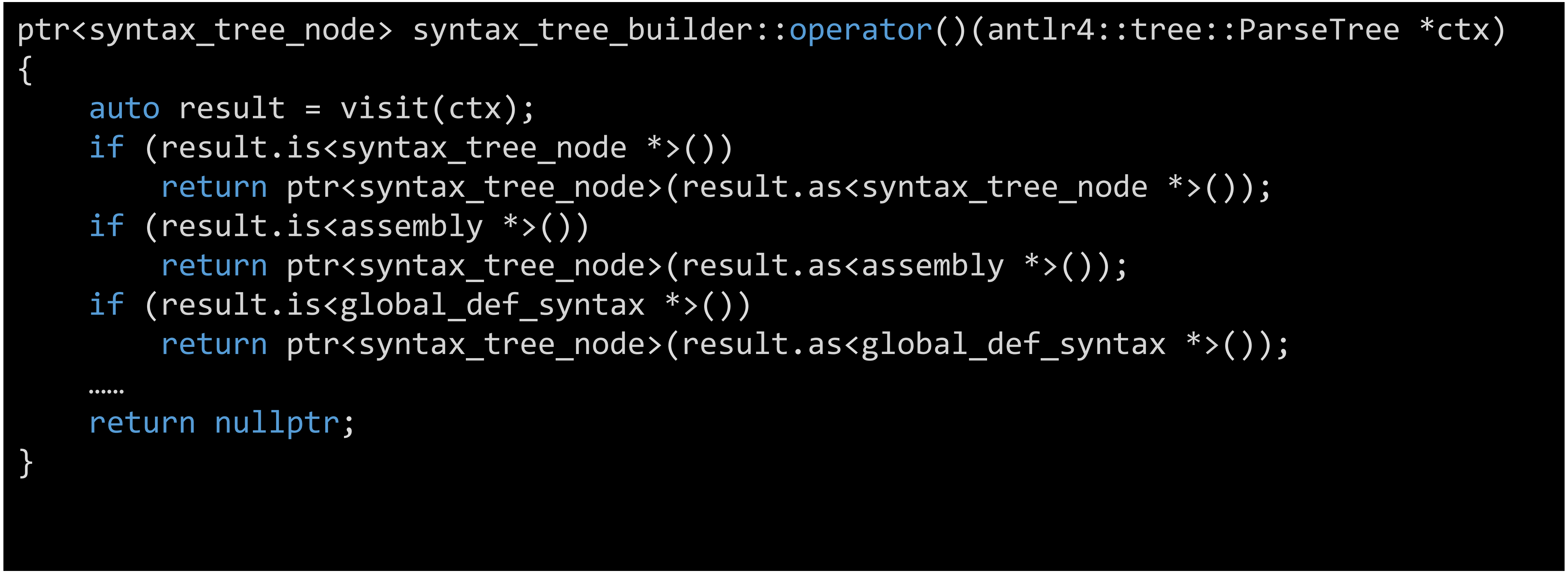
Lab：ParseTree =>AST

- □ 由解析树构造AST c1recognizer/src/syntax_tree_builder.cpp

```
antlrcpp::Any syntax_tree_builder::visitNumber(C1Parser::NumberContext *ctx)
{
    auto result = new literal_syntax;
    if (auto intConst = ctx->IntConst())
    {
        result->is_int = true;
        result->line = intConst->getSymbol()->getLine();
        result->pos = intConst->getSymbol()->getCharPositionInLine();
        auto text = intConst->getSymbol()->getText();
        if (text[0] == '0' && (text[1] == 'x' || text[1] == 'X')) // Hexadecimal
            result->intConst = std::stoi(text, nullptr, 16); // std::stoi will eat '0x'
        /* you need to add other situations here */

        return static_cast<expr_syntax *>(result);
    }
    // else FloatConst
    else
    {
        return static_cast<expr_syntax *>(result);
    }
}
```

Lab：ParseTree =>AST

- □ 由解析树构造AST c1recognizer/src/syntax_tree_builder.cpp



```
ptr<syntax_tree_node> syntax_tree_builder::operator()(antlr4::tree::ParseTree *ctx)
{
    auto result = visit(ctx);
    if (result.is<syntax_tree_node *>())
        return ptr<syntax_tree_node>(result.as<syntax_tree_node *>());
    if (result.is<assembly *>())
        return ptr<syntax_tree_node>(result.as<assembly *>());
    if (result.is<global_def_syntax *>())
        return ptr<syntax_tree_node>(result.as<global_def_syntax *>());
    ……
    return nullptr;
}
```

Lab：ParseTree =>AST

□ AST的定义

c1recognizer/include/c1recognizer/syntax_tree.h

■ 引用对象的类型及其列表类型

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

■ Node

```cpp
struct syntax_tree_node
{
    int line;
    int pos;
    // Used in syntax_tree_visitor.
    virtual void accept(syntax_tree_visitor
               &visitor) = 0;
};
```

https://www.educoder.net/shixuns/qix6mfn3/challenges

```cpp
struct syntax_tree_node;
struct assembly : syntax_tree_node;
struct global_def_syntax : virtual syntax_tree_node;
    struct func_def_syntax : global_def_syntax;
struct cond_syntax : syntax_tree_node;

struct expr_syntax : virtual syntax_tree_node;
    struct binop_expr_syntax : expr_syntax;
    struct unaryop_expr_syntax : expr_syntax;
    struct lval_syntax : expr_syntax;
    struct literal_syntax : expr_syntax;

struct stmt_syntax : virtual syntax_tree_node;
    struct var_def_stmt_syntax : stmt_syntax,
    global_def_syntax;
    struct assign_stmt_syntax : stmt_syntax;
    struct func_call_stmt_syntax : stmt_syntax;
    struct block_syntax : stmt_syntax;
    struct if_stmt_syntax : stmt_syntax;
    struct while_stmt_syntax : stmt_syntax;
    struct empty_stmt_syntax : stmt_syntax;
```

Lab：ParseTree =>AST

□ AST的访问者Visitor

c1recognizer/include/c1recognizer/syntax_tree.h

```
class syntax_tree_visitor {
  public:
    virtual void visit(assembly &node) = 0;
    virtual void visit(func_def_syntax &node) = 0;
    virtual void visit(cond_syntax &node) = 0;
    virtual void visit(binop_expr_syntax &node) = 0;
    virtual void visit(unaryop_expr_syntax &node) = 0;
    virtual void visit(lval_syntax &node) = 0;
    virtual void visit(literal_syntax &node) = 0;
    virtual void visit(var_def_stmt_syntax &node) = 0;
    virtual void visit(assign_stmt_syntax &node) = 0;
    virtual void visit(func_call_stmt_syntax &node) = 0;
    virtual void visit(block_syntax &node) = 0;
    virtual void visit(if_stmt_syntax &node) = 0;
    virtual void visit(while_stmt_syntax &node) = 0;
    virtual void visit(empty_stmt_syntax &node) = 0;
};
```

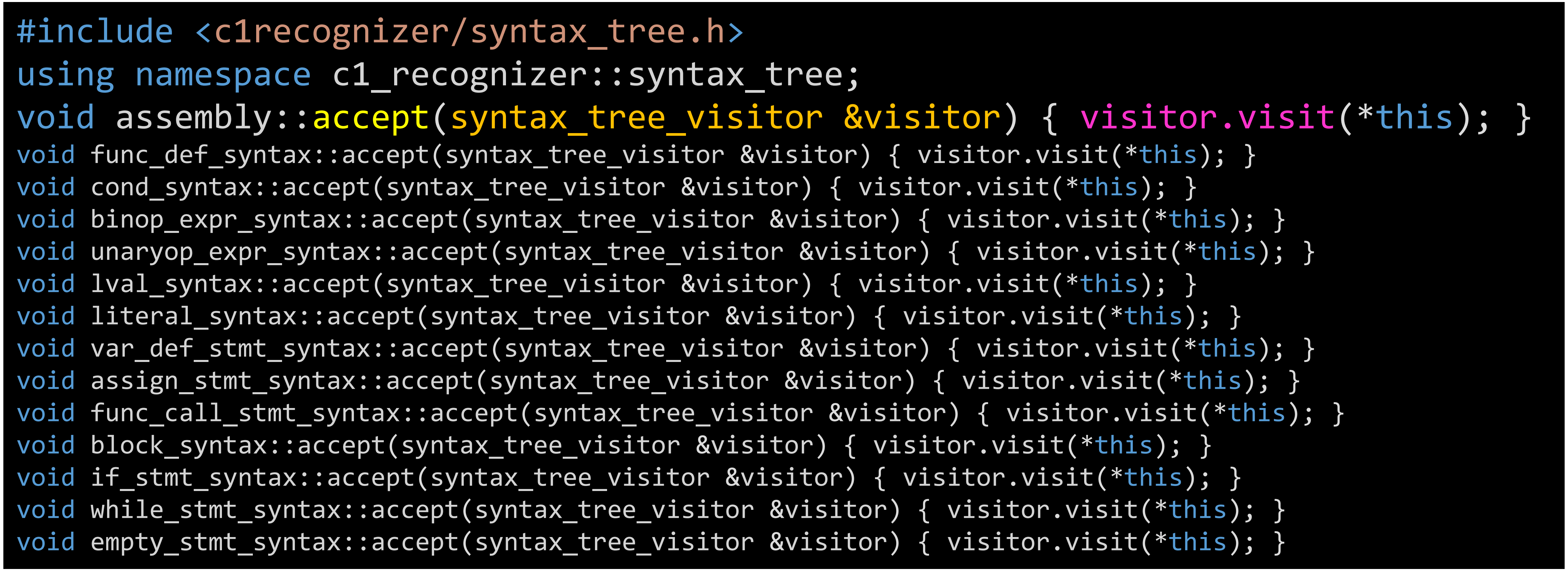
```
struct syntax_tree_node;
struct assembly : syntax_tree_node;
struct global_def_syntax : virtual syntax_tree_node;
    struct func_def_syntax : global_def_syntax;
struct cond_syntax : syntax_tree_node;

struct expr_syntax : virtual syntax_tree_node;
    struct binop_expr_syntax : expr_syntax;
    struct unaryop_expr_syntax : expr_syntax;
    struct lval_syntax : expr_syntax;
    struct literal_syntax : expr_syntax;

struct stmt_syntax : virtual syntax_tree_node;
    struct var_def_stmt_syntax : stmt_syntax,
    global_def_syntax;
    struct assign_stmt_syntax : stmt_syntax;
    struct func_call_stmt_syntax : stmt_syntax;
    struct block_syntax : stmt_syntax;
    struct if_stmt_syntax : stmt_syntax;
    struct while_stmt_syntax : stmt_syntax;
    struct empty_stmt_syntax : stmt_syntax;
```

https://www.educoder.net/shixuns/qix6mfn3/challenges

Lab：ParseTree =>AST

□ AST结点的访问：通过访问者来访问结点

c1recognizer/src/syntax_tree.cpp



```
#include <c1recognizer/syntax_tree.h>
using namespace c1_recognizer::syntax_tree;
void assembly::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void func_def_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void cond_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void binop_expr_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void unaryop_expr_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void lval_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void literal_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void var_def_stmt_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void assign_stmt_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void func_call_stmt_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void block_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void if_stmt_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void while_stmt_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
void empty_stmt_syntax::accept(syntax_tree_visitor &visitor) { visitor.visit(*this); }
```

https://www.educoder.net/shixuns/qix6mfn3/challenges