Lab1: 词法与语法分析

Author: wzj Date: 2025.3.22

实现功能

Sysy语言的词法分析

参考实验手册的App. A: SysY 语言规范中SysY语言的文法表示， 可以得到需要补充的终结符，在 lexel.l 和 parser.y 中进行添加即可

Sysy语言的语法分析

同样，参考实验手册的App. A: SysY 语言规范中SysY语言的文法表示， 在 parser.y 中添加需要补充的非终结符，并将其文法表示改写为符合 bison 要求的 rules。

其中， 文法表示中符号 [...] 表示方括号内包含的项可被重复 0 次或 1 次, 在改写为规则时可以使用枚举两种情况的方法

文法表示中符号 {...} 表示花括号内包含的项可被重复 0 次或多次， 实现“多次”，可以在改写为规则时使用递归定义, 如下图，VarDefs即可表示多个变量的定义

```
VarDefs : VarDef { $$ = new VarDecl(shared_cast<VarDef>($1)); }
    | VarDefs "," VarDef { static_cast<VarDecl*>($1)-
>add_def(shared_cast<VarDef>($3)); $$ = $1; }
    ;
```

生成打印完整的AST

由于新添加的规则，我们需要定义更多的AST节点类型，本实验中，我补充了 EmptyStmt, IfStmt, WhileStmt, FuncFParams等节点定义，并为重要节点编写了相应的to_string()方法，使print_tree()能正确执行打印完整的AST，并且记录了重要信息

技术亮点

本次实验较为简单，我完成的也直白，很难说有什么独创性的构思，但在为了支持数组定义、初始化列表以及函数参数记录时，额外需要一些代码改动：

为支持数组定义，需要为VarDef添加dim属性和新的初始化函数和to_string函数：

```
class VarDef : public Node {
 public:
  std::vector<int> dim;
  VarDef(VarDefPtr var, int d) : ident(var->ident), dim(var->dim) {
add_dim(d); }
  void add_dim (int d) { dim.push_back(d); }
  std::string to_string() override {
```

```
    if (dim.size() > 0) {
      std::string dim_str = "dim: (";
      for (int d : dim) {
        dim_str += std::to_string(d) + ",";
      }
      dim_str.pop_back();
      dim_str += ")";
      return "VarDef <ident: " + ident + ", " + dim_str + ">";
    }
    return "VarDef <ident: " + ident + ">";
  }
};
```

为支持初始化以及在AST中记录初始化的数据，我设计了新的节点类型InitList和InitElements, 并将InitList作为VarDef的children, 补充前者的to_string函数和后者的属性、初始化函数以及get_children函数

```
class VarDef : public Node {
 public:
  // add init list
  InitListPtr inits;

  VarDef(char const *ident, InitListPtr inits) : ident(ident), inits(inits)
{};
  VarDef(VarDefPtr var, int d, InitListPtr inits) : ident(var->ident),
dim(var->dim), inits(inits) { add_dim(d); }

  std::vector<NodePtr> get_children() override {
    if (inits) {
      return {inits};
    }
    return {};
  }
};

class InitElements : public Node {
 public:
  std::vector<NodePtr> elements;
  InitElements(NodePtr element) { add_element(element); }
  void add_element(NodePtr element) { elements.push_back(element); }
  std::string to_string() override { return "InitElements"; }
  std::vector<NodePtr> get_children() override { return elements; }
};

class InitList : public Node {
 public:
  std::vector<NodePtr> inits;
  InitList() {}
  InitList(NodePtr init) { add_init(init); }
  void add_init(NodePtr init) { inits.push_back(init); }
  std::string to_string() override { return "InitList"; }
```

```
  std::vector<NodePtr> get_children() override { return inits; }
};
```

为支持函数数组参数，我设计了一个“中间量”非终结符ArrayDims, 用于记录维度

```
ArrayDims : "[" INTCONST "]" { $$ = new ArrayDims($2); }
    | ArrayDims "[" INTCONST "]" { static_cast<ArrayDims*>($1)-
>add_dim($3); $$ = $1; }
    ;

FuncFParam : "int" IDENT { $$ = new FuncFParam($2); }
    | "int" IDENT "[" "]" { $$ = new FuncFParam($2); }
    | "int" IDENT "[" "]" ArrayDims { $$ = new FuncFParam($2,
shared_cast<ArrayDims>($5)); }
    ;
```

为支持在AST中记录函数参数，需要设计一个FuncFParams（略）并将其加入上层节点FuncDef的children, 并为前者补充to_string()方法，并修改后者的get_children方法

```
class FuncDef : public Node {
    // to support params:
  FuncFParamsPtr params;

  FuncDef(BasicType return_btype, char const *name, BlockPtr block)
      : return_btype(return_btype), name(name), block(block) {}

  FuncDef(BasicType return_btype, char const *name, BlockPtr block,
FuncFParamsPtr params)
      : return_btype(return_btype), name(name), block(block),
params(params) {}

  std::vector<NodePtr> get_children() override {
    if (params) {
      return {params, block};
    }
    return {block};
  }
};
```

Reference

1. 实验过程中使用copilot-chat/chatGPT, 用于分析一些报错信息以及查询知识空缺
2. 阅读NJU的C语言词法分析器和语法分析器，**无复用代码**

实验设计建议

建议添加一个 General Test, 能够对该文件进行语法分析即可说明语法/词法功能实现正确，与AST的部分解耦。