Compiler Design
编 译 器 构 造 实 验

Lab 8: Project 3

张献伟

xianweiz.github.io

DCS292, 4/22/2021

总体任务

- 变量绑定: 将变量的使用和定义绑定
  - 截止到project2: 变量绑定还从未发生
    - Identifiers nodes just point to entries in the string table
  - project 3之后: 所有变量完成绑定
    - All the identifier nodes in the AST would have been replaced by symbol table nodes that point to entries in the symbol table
    - Symbol table entry: a definition the identifier, and all relevant info

- 同时, 执行语义检查
  - 是在完成语法分析后再通过遍历树来分析语义（并不是在语法分析过程中）
    - Example: whether identifiers are used without being defined
  - 并不需要自己实现符号表的所有操作
    - Call the provided functions in proj3.c

代码结构

- Makefile
- From project 2:
  - lexer.l   Use your own implementation
    - For lexical analysis, providing tokens
  - grammar.y   Use your own implementation
    - For syntax parsing, providing abstract syntax tree
  - proj2.[h|c]   Use your own 'loc_str()'
    - Tree manipulation routines (mostly unchanged)
- For project 3:
  - semantic.c
    - Primary place to write your codes
  - proj3.[h|c]
    - Symbol table manipulation routines (mostly unchanged)

主要流程

- Traverse parse tree after parsing
  - Traver the parse tree to perform semantic actions

```
Program : PROGRAMnum ID SEMInum ClassDecl1
    { $$ = MakeTree(ProgramOp, $4, $2); printtree($$, 0); }
    ;
```

grammar.y

```
FILE *treelst;
main() {
   treelst = stdout;
   yyparse();
}
```

```
parseTree = $$;
```

grammar.y

```
FILE *treelst;
main() {
    treelst = stdout;
    yyparse();
    do_semantic(parseTree);  // Do semantic analysis
    printtree(parseTree, 0);    // Print the parse tree
}
```

semantic.c

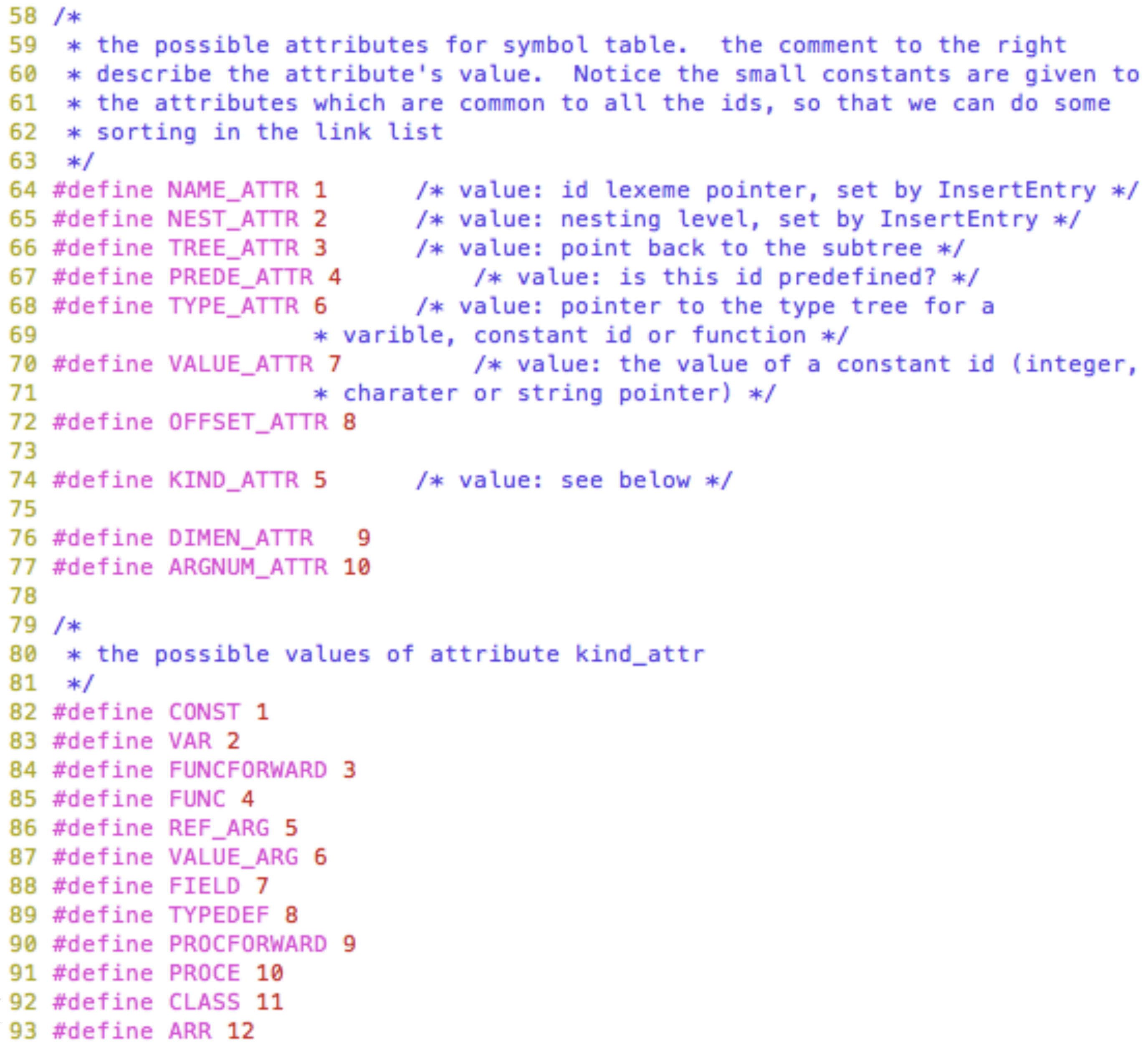
```
void do_semantic(tree parseTree) {
     STInit();                    // Initialize the symbol table
     traverse(parseTree);         // Traverse tree
     STPrint();                   // Print the symbol table
}
```

APIs of Symbol Table [proj3.c]

- Proj3.c contains an implementation of a symbol table using stack
  - What to do? Call the correct APIs to generate the symbol table using that implementation

```
STInit()
InsertEntry(ID:integer) return STIndex
LookUP(ID:integer) return STIndex
LoopUpHere(ID:integer) return STIndex
LookUPField(ID:integer) return STIndex
OpenBlock()
CloseBlock()
IsAttr(ST:STIndex; AttrNum:integer) return boolean
GetAttr(ST:STIndex; AttrNum: integer) return integer, boolean or ILTree
SetAttr(ST:STIndex; AttrNum: integer; V: integer, boolean or ILTree)
STPrint()
```

Symbol Attributes [proj3.h]

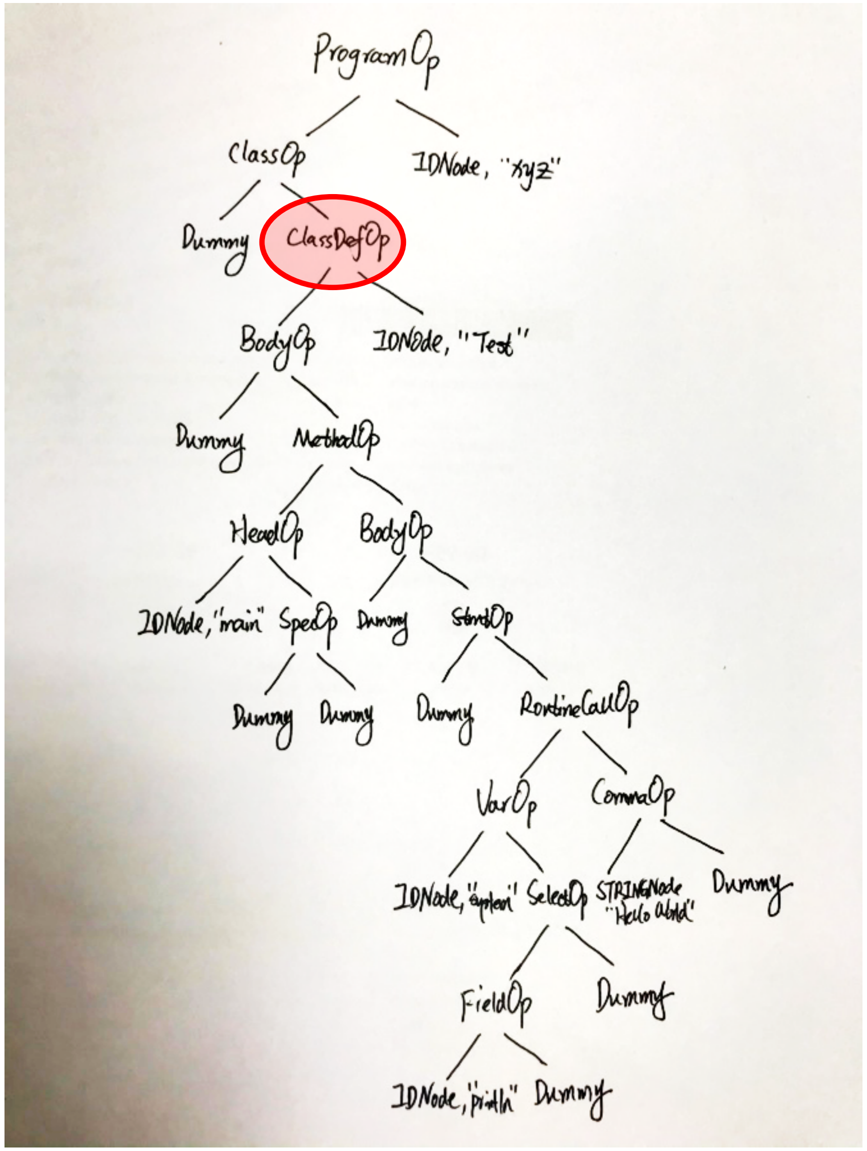


```c
/*
 * the possible attributes for symbol table.  the comment to the right
 * describe the attribute's value.  Notice the small constants are given to
 * the attributes which are common to all the ids, so that we can do some
 * sorting in the link list
 */
#define NAME_ATTR 1      /* value: id lexeme pointer, set by InsertEntry */
#define NEST_ATTR 2      /* value: nesting level, set by InsertEntry */
#define TREE_ATTR 3      /* value: point back to the subtree */
#define PREDE_ATTR 4         /* value: is this id predefined? */
#define TYPE_ATTR 6      /* value: pointer to the type tree for a
                  * varible, constant id or function */
#define VALUE_ATTR 7         /* value: the value of a constant id (integer,
                  * charater or string pointer) */
#define OFFSET_ATTR 8

#define KIND_ATTR 5      /* value: see below */

#define DIMEN_ATTR   9
#define ARGNUM_ATTR 10

/*
 * the possible values of attribute kind_attr
 */
#define CONST 1
#define VAR 2
#define FUNCFORWARD 3
#define FUNC 4
#define REF_ARG 5
#define VALUE_ARG 6
#define FIELD 7
#define TYPEDEF 8
#define PROCFORWARD 9
#define PROCE 10
#define CLASS 11
#define ARR 12
```

STInit()

```
void
STInit()
{
  int nStrInd, nSymInd;  /* string table index */

  nStrInd = loc_str("system"); /* return string index of string "system" */
  if ( nStrInd != -1 )         /* "system" is stored in string table */
    {
      nSymInd = InsertEntry(nStrInd);
      /* SetAttr(nSymInd, TREE_ATTR, NULL); */
      SetAttr(nSymInd, PREDE_ATTR, true);
      SetAttr(nSymInd, KIND_ATTR, CLASS);
    }

  nStrInd = loc_str("readln");
  if ( nStrInd != -1 )
    {
      nSymInd = InsertEntry(nStrInd);
      SetAttr(nSymInd, NEST_ATTR, nesting+1);
      SetAttr(nSymInd, ARGNUM_ATTR, 1);
      /* SetAttr(nSymInd, TREE_ATTR, NULL); */
      SetAttr(nSymInd, PREDE_ATTR, true);
      SetAttr(nSymInd, KIND_ATTR, PROCE);
    }

  nStrInd = loc_str("println");
  if ( nStrInd != -1 )
    {
      nSymInd = InsertEntry(nStrInd);
      SetAttr(nSymInd, NEST_ATTR, nesting+1);
      SetAttr(nSymInd, ARGNUM_ATTR, 1);
      /* SetAttr(nSymInd, TREE_ATTR, NULL); */
      SetAttr(nSymInd, PREDE_ATTR, true);
      SetAttr(nSymInd, KIND_ATTR, PROCE);
    }

}
```

Example



IDNode --> STNode
Point to symbol table entry

```
void traverseClassDefOp (tree startNode) {
   int nStrInd, nSymInd, tmp;

   nStrInd = loc_str(getname(startNode->RightC->IntVal));
   nSymInd = InsertEntry(nStrInd);
   tmp = startNode->RightC->IntVal;
   if (nSymInd != 0) {
      startNode->RightC->NodeKind = STNode;
      startNode->RightC->IntVal = nSymInd;
      startNode->RightC->NodeOpType = tmp;
   }

   SetAttr(nSymInd, KIND_ATTR, CLASS);
   OpenBlock ();
   traverseInClassBody (startNode->LeftC);
   CloseBlock ();
}
```

关于评分

得分项	分数
1. 程序编写 (25%)	
(1) 结构，注释，清晰度 (15%)	
(2) 结果输出 (10%)	
2. 功能实现 (75%)	
(1) Basics: Tree traverse, nesting level, STNode (7.5%)	
(2) Name declaration uniqueness in a method (7.5%)	
(3) Name declaration uniqueness in a block (7.5%)	
(4) Method/variable visibility (7.5%)	
(5) Class method/variable visibility (7.5%)	
(6) Nested class variable visibility (7.5%)	
(7) Array type check on index operation (7.5%)	
(8) Additional functionality 1/6 (7.5%)	
(9) Additional functionality 2/6 (7.5%)	
(10) Additional functionality 3/6 (7.5%)	
总计（100分）	

```
void func();
func();

Point p;
p.func();
p.x = 10;
p.y = 20;

p0.p1.x = 10;
p0.p1.y = 20;
```