编译原理

课程Project说明

COMP130014.01

2022.09

简介

- 本学期共有两个Project：
- Project 1：词法分析，占比**40%**
- Project 2：语法分析，占比**60%**
- TA联系方式：
  - 何瑞安 rahe21@m.fudan.edu.cn (负责Project)
  - 谈天 tant21@m.fudan.edu.cn (负责课程作业)

实验环境

- OS：Linux
- 依赖：gcc/g++, make，flex, bison
- gcc，make，flex与bison安装(以Ubuntu为例)：
  - sudo apt-get install build-essential flex bison
- 实验环境也可自行在MAC OS以及WINDOWS下配置，建议使用虚拟机安装Ubuntu。
- 简单来说，就是C/C++配合flex与bison两个工具完成实验

实验目的

- 通过flex与bison，分析目标PCAT语言，并**生成目标语言的语法树**
- PCAT语言可看作一种简化版的PASCAL语言：

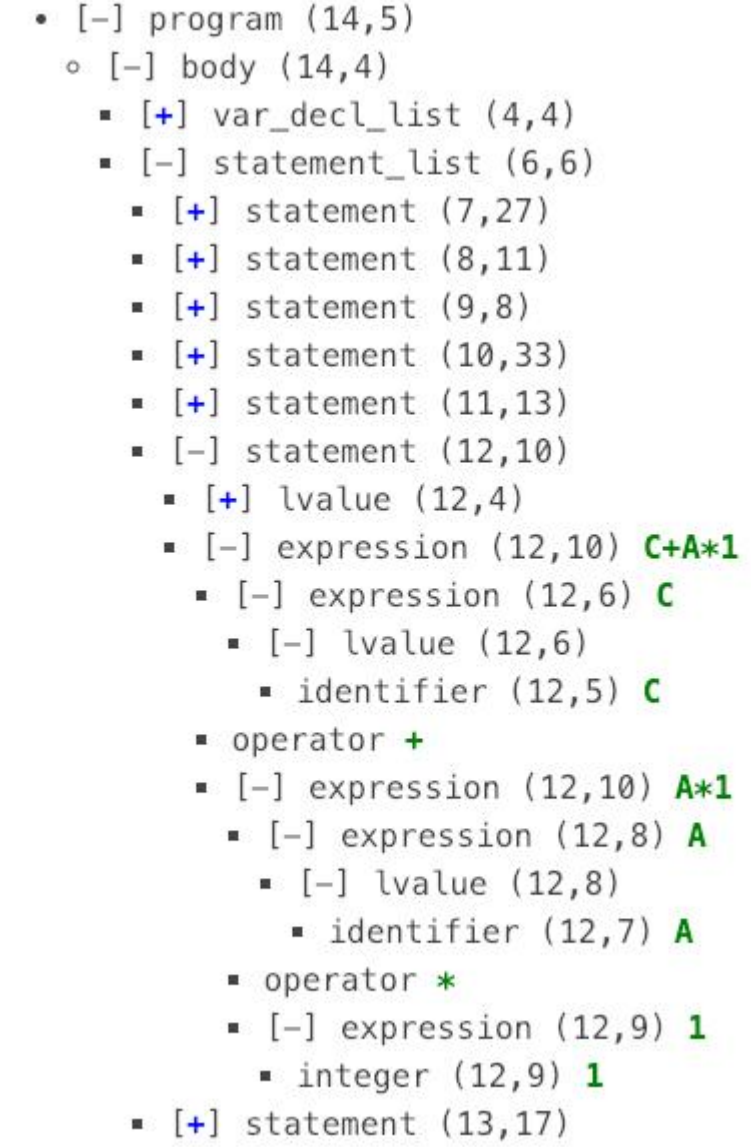
```
PROGRAM IS
    VAR i, j : INTEGER := 1;
    VAR x : REAL := 2.0;
    VAR y : REAL := 3.0;
BEGIN
    WRITE ("i = ", i, ", j = ", j);
    WRITE ("x = ", x, ", y = ", y);
END;
```

Project 2 语法分析 (60%)

- 任务：结合Project 1完成的词法分析，使用flex & bison建立PCAT语言的语法树

```
(* Testing reals. *)

PROGRAM IS
    VAR A, B : REAL := 0.0;
    C : REAL := 0.0;
BEGIN
    WRITE ("ENTER TWO REALS:");
    READ (A, B);
    C := 8.0;
    WRITE ("A=", A, ", B=", B, ", C=", C);
    C := A - (-B) + C;
    C := C + A * 1;
    WRITE ( -C /(A + 1));
END;
```



Bison介绍

- Bison是一个通用的解析器生成器，它将LALR(1)上下文无关语法的语法描述（.y）转换为C程序（.c）来解析该语法。
- .y文件的结构布局：

```
%{
C declarations //声明语法树结构Node，声明语法树的相关函数，或者include其他文件
%}

Bison declarations//声明终结符（token）和非终结符（type），以及其返回值

%%
Grammar rules//主要的语法解析步骤，自下而上构建语法树，createNode
%%
Additional C code//进行语法树的打印，构建等相关函数的实现，printNode，createNode
```

Demo：实现一个简单的计算器

- 表达式的文法可以写成：

```
S -> empty | S E \n
E -> F | E + F | E – F
F -> T | F * T | F / T
T -> number | ( E )
```

- 请写出句型(2+3)*5+5\n的规范规约及每一步的句柄，给出“移进-规约”的过程并构造语法树。

- 回忆曾经学过的“中缀表达式”、“后缀表达式”，联系算法优先分析法，思考程序语言使用LR分析法的好处。

实现一个简单的计算器

- 首先找到文法中的终结符tokens
- Tokens:
  - +, -, *, /, (,), number, \n

- 需要在Bison中定义返回值和终结符（token），非终结符（type）
- 识别token的部分已经在lexer实现

- 编写过程： bison -> flex -> main

```
S -> empty | S E \n
E -> F | E + F | E – F
F -> T | F * T | F / T
T -> number | ( E )
```

demo.y

```
%{
#include <iostream>
using namespace std;
#include "lex.c"//从lexer中导入TOKEN
%}

%union {
   double val;  //定义返回值yylval.val为double
}
%token <val> NUMBER //定义NUMBER具有val属性
%token ADD SUB MUL DIV OP CP
%token EOL

%type <val> exp // exp为非终结符，返回val属性
%type <val> factor
%type <val> term
```

实现一个简单的计算器

- Bison语法

  左部符号:右部{语义动作};

  右部各侯选式用| 分割
- $$表示和左部非终结符相关的属性值，会传回上层
- $k表示和右部第k个文法相关的属性值，来自下层传回（这里既包含终结符也包含非终结符）
- 语义动作表示左部直接等于右部时

  即{$$=$1;}可以忽略不写。
- 语义动作也可以返回函数值，执行C函数，也可以多行（;号分割），可以用来自下而上构造语法树树形结构。

```
S -> empty | S E \n
E -> F | E + F | E – F
F -> T | F * T | F / T
T -> number | ( E )
```

demo.y

```
%%
calc:
  | calc exp EOL { cout << "= " << $2 << endl; }
  ;
exp: factor
  | exp ADD factor { $$ = $1 + $3; }
  | exp SUB factor { $$ = $1 - $3; }
  ;
factor: term
  | factor MUL term { $$ = $1 * $3; }
  | factor DIV term { $$ = $1 / $3; }
  ;
term: NUMBER
  | OP exp CP { $$ = $2; }
  ;
%%
```

实现一个简单的计算器

- 使用bison编译demo.y文件： bison -o yacc.c -d demo.y
- 产生两个文件：yacc.c / yacc.h
- 请仔细观察yacc.c生成规律并阅读bison文档
- demo.lex:

```
%{
#include "yacc.h"
%}
%option    nounput
%option    noyywrap

DIGIT      [0-9]
INTEGER    {DIGIT}+
REAL       {DIGIT}+"."{DIGIT}*
WS         [ \t]+
```

```
%%
{WS}       /* skip blanks and tabs */
"+"        return ADD;
"-"        return SUB;
"*"        return MUL;
"/"        return DIV;
"("        return OP;
")"        return CP;
\n         return EOL;
{INTEGER}|{REAL}   { yylval.val = atof(yytext);
return NUMBER; }
%%
```

实现一个简单的计算器

- 使用flex编译demo.lex文件：
- flex -o lex.c demo.lex
- 编写main函数(demo.cpp)：
- 使用g++编译C/C++文件：
  - g++ -c yacc.c
  - g++ demo.cpp yacc.o -o demo

```
int yyparse();
extern "C" FILE* yyin;

int main(int argc, char* args[]) {
  if (argc > 1) {
    FILE *file = fopen(args[1], "r");
    if (!file) {
      cerr << "Can not open file." << endl;
      return 1;
    } else {
      yyin = file;
    }
  }

  yyparse();
  return 0;
}
```

参考资料

- 下发文件PCAT语言参考PDF中有相应的LL1文法参考，实现以该说明为准。

- Flex manual:

http://ranger.uta.edu/~fegaras/cse5317/flex/flex_toc.html

- Bison manual:

http://ranger.uta.edu/~fegaras/cse5317/bison/bison_toc.html

参考资料

12 Complete Concrete Syntax

```
program          -> PROGRAM IS body ';'
body             -> {declaration} BEGIN {statement} END
declaration      -> VAR {var-decl}
                 -> TYPE {type-decl}
                 -> PROCEDURE {procedure-decl}
var-decl         -> ID { ',' ID } [ ':' type ] ':=' expression ';'
type-decl        -> ID IS type ';'
procedure-decl   -> ID formal-params [':' type] IS body ';'
type             -> ID
                 -> ARRAY OF type
                 -> RECORD component {component} END
component        -> ID ':' type ';'
formal-params    -> '(' fp-section {';' fp-section } ')'
                 -> '(' ')'
fp-section       -> ID {',' ID} ':' type
statement        -> lvalue ':=' expression ';'
                 -> ID actual-params ';'
                 -> READ '(' lvalue {',' lvalue} ')' ';'
                 -> WRITE write-params ';'
                 -> IF expression THEN {statement}
                    {ELSIF expression THEN {statement}}
                    [ELSE {statement}] END ';'
                 -> WHILE expression DO {statement} END ';'
                 -> LOOP {statement} END ';'
                 -> FOR ID ':=' expression TO expression [ BY expression ] DO {statement} END ';'
                 -> EXIT ';'
                 -> RETURN [expression] ';'
```

参考资料

```
write-params    -> '(' write-expr {',' write-expr } ')'
                -> '(' ')'
write-expr      -> STRING
                -> expression
expression      -> number
                -> l-value
                -> '(' expression ')'
                -> unary-op expression
                -> expression binary-op expression
                -> ID actual-params
                -> ID comp-values
                -> ID array-values
l-value         -> ID
                -> l-value '[' expression ']'
                -> l-value '.' ID
actual-params   -> '(' expression {',' expression} ')'
                -> '(' ')'
comp-values     -> '{' ID ':=' expression { ';' ID ':=' expression} '}'
array-values    -> '[<' array-value { ',' array-value } '>]'
array-value     -> [ expression 'OF' ] expression
number          -> INTEGER | REAL
unary-op        -> '+' | '-' | NOT
binary-op       -> '+' | '-' |  '*' | '/' | DIV | MOD | OR | AND
                -> '>' | '<' | '=' | '>=' | '<=' | '<>'
```

Project 2 评分细则

项目完成度及正确性：（共计60分）

1. 正确分析case 1-10的语法，并打印出语法树，语法树的输出形式可自定义，但必须要能看出树形结构**（必做，40分）**

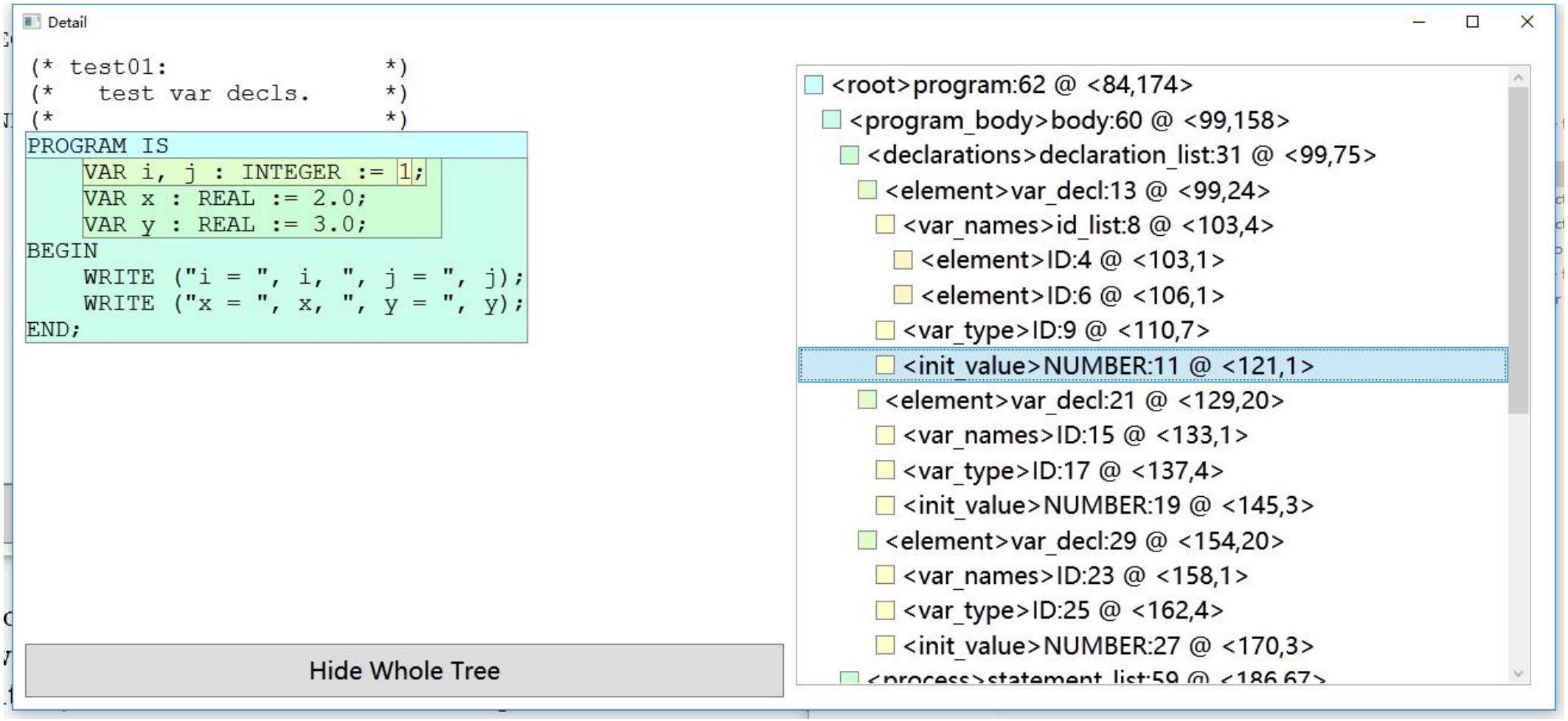
2. 正确分析case 11-14中出现的各种词法错误和语法错误，提供相应报错信息并提示错误的位置（选做，20分）

项目报告及展示：（共计40分）

1. 撰写项目报告，说明项目的文件组织结构，bison的用法，各项语法规则的实现，词法及语法错误检测，语法树实现，makefile实现等等，在结尾标明分工及贡献百分比**（必做，20分）**

2. 项目报告完成后，与TA预约，在上机课时间向TA展示样例的语法分析结果，TA会就项目相关内容进行简单的提问**（必做，20分）**

Project 2 语法树优秀展示



Project 2 语法树优秀展示

```
<!-- item template -->
<script type="text/x-template" id="item-template">
    <li>
    <div v-if="isError">
        {{ model.error}}
    </div>
    <div
        :class="{bold: isFolder}"
        @click="toggle">

        <span v-if="isFolder">[{{ open ? '-' : ''}}<b style="color:blue">{{ open
        ? '' : '+'}}</b>]</span>
        {{ model.name }}
        <span v-if="model.line!=0">
            ({{ model.line }},{{ model.col }})
        </span>
        <b style="color:green"> {{model.info}} </b>
    </div>
    <ul v-show="open" v-if="isFolder">
        <item
        class="item"
        v-for="(model, index) in model.children"
        :key="index"
        :model="model">
        </item>
    </ul>
    </li>
```

- [−] program (14,5)
  - [−] body (14,4)
    - [+] var_decl_list (4,4)
    - [−] statement_list (6,6)
      - [+] statement (7,27)
      - [+] statement (8,11)
      - [+] statement (9,8)
      - [+] statement (10,33)
      - [+] statement (11,13)
      - [−] statement (12,10)
        - [+] lvalue (12,4)
        - [−] expression (12,10) **C+A*1**
          - [−] expression (12,6) **C**
            - [−] lvalue (12,6)
              - identifier (12,5) **C**
          - operator **+**
          - [−] expression (12,10) **A*1**
            - [−] expression (12,8) **A**
              - [−] lvalue (12,8)
                - identifier (12,7) **A**
            - operator *****
            - [−] expression (12,9) **1**
              - integer (12,9) **1**
      - [+] statement (13,17)

Project 2 提交方式

- 项目代码，运行结果（txt或其他）及项目报告（PDF）请打包（zip）并发送至TA邮箱 rahe21@m.fudan.edu.cn
- 邮件/压缩包标题：2022编译原理PJ2 姓名1 姓名2
- **项目报告DDL: 2022年12月16日 23:59**
- 提交报告后与助教预约展示时间
- **展示DDL: 2021年12月23日上机课**

- 如果文件太大，可先上传至百度云或者复旦云，再将网盘分享地址发送到TA邮箱
- 若对Project有疑问，或想在上机课外时间展示，可与TA联系
- TA办公地址：江湾校区交叉学科2号楼A4008室

- **严禁抄袭，包括网络上和同学的代码，一经发现Project作0分处理**
- **只实现必做功能也一定可以顺利通过，不要铤而走险**