编译Project 2

COMP130014.01

2019.11

简介

- Project分成两部分，评分分别占40%与60%
- 每组不超过4个人
- 小组成员名单发送至负责PJ的TA
- Project介绍及demo，demo基础上补充完成相应功能即可：
  https://github.com/WangJY06/CompilePJ
- TA联系方式：
  - 王峻逸 18110240005@fudan.edu.cn (负责Project)
  - 林青　18110240028@fudan.edu.cn (负责课程作业)

实验环境

- OS：linux，推荐Ubuntu
- 依赖：gcc/g++ (版本不限)，flex, bison
- flex与bison安装(以Ubuntu为例)：
  - sudo apt-get install flex
  - sudo apt-get install bison
- 实验环境也可在MAC OS以及WINDOWS下配置，具体配置请自行搜索
- 简单来说，就是C/C++配合flex与bison两个工具

实验目的

- 通过flex与bison两种工具，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

```
(* test01:              *)
(*   test var decls.    *)
(*                      *)
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 (60%)

- Project介绍及demo：
  https://github.com/WangJY06/CompilePJ

- 结合之前完成的内容，使用flex & bison完成对PCAT语言的语法树建立
- 完成基本功能，打印出语法树，**树结构请自行构建**（30%）
- 具有语法报错功能并提示错误位置，**请自行修改测试文件，在验收和报告中举例说明**（10%）
- 编写文档，包括bison的使用方法，实现的细节等等，最后说明项目的成员分工及贡献百分比**(全25%则平分分数衰减，1-2人小组不衰减)**（20%）

- 提交项目代码以及项目文档PDF
- DDL: **期末考试前最后一次上机**
- 该项目需展示给TA，先完成可先展示，时间与TA预约即可

Project 2 示例

- 输出格式参考，能看出是树形结构即可

Parse Tree of test08.pcat

```
1. [-] program (14,5)
  1. [-] body (14,4)
    1. [-] declaration_list (3,10)
      1. [-] procedure_decl (9,5)
        1. identifier (4,13) FOO
        2. [-] component (4,27)
          1. [-] fp_section (4,19)
            1. identifier (4,15) X
            2. identifier (4,19) INT
          2. [-] fp_section (4,26)
            1. identifier (4,21) Y
            2. identifier (4,26) REAL
        3. identifier (4,32) REAL
        4. [-] body (9,4)
          1. [-] procedure_decl (5,35)
            1. identifier (5,13) BAR
            2. formal_params (5,15)
            3. typename (5,17)
            4. [-] body (5,34)
              1. [-] statement (5,31)
                1. [-] lvalue (5,25)
                  1. identifier (5,23) Y
                2. [-] expression (5,31) X+1.0
                  1. [-] expression (5,27) X
                    1. [-] lvalue (5,27)
                      1. identifier (5,26) X
                  2. operator +
                  3. [-] expression (5,30) 1.0
                    1. real (5,30) 1.0
```

提交方式

- 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；文件小则可直接发送到TA邮箱。
- TA邮箱：18110240005@fudan.edu.cn

- 若对PJ有疑问，或想在上机课外时间验收，可与TA联系
- TA办公地址：新金博大厦1201室（邯郸路539号）

词法分析：回顾

- 目标：提取出PCAT语言的Tokens
- 第一类：关键词（keywords）以及符号（operator & delimiter)

```
AND       ARRAY     BEGIN     BY        DIV       DO        ELSE
ELSIF     END       EXIT      FOR       IF        IN        IS
LOOP      MOD       NOT       OF        OR        OUT       PROCEDURE
PROGRAM   READ      RECORD    RETURN    THEN      TO        TYPE
VAR       WHILE     WRITE

operator  = ":="|'+'|'-'|'*'|'/'|'<'|"<="|'>'|">="|'='|"<>"
delimiter = ':'|';'|','|'.'|'('|')'|'['|']'|'{'|'}'|"[<"|">]"|'\'
```

词法分析：回顾

- 第二类：整型/实数型/字符串/变量名

```
.lex:
DIGIT           [0-9]
LETTER          [a-zA-Z]
INTEGER         {DIGIT}+
REAL            {DIGIT}+"."{DIGIT}*
STRING          \“([^\t\n”])*\“                 // 字符串中不包含制表符/回车符/引号
ID              {LETTER}({LETTER}|{DIGIT})*     // 变量名必须以字符开头

%%

{INTEGER}       return integer_check();
{REAL}          return T_REAL;
{STRING}        return length_check(T_STRING, MAX_LEN + 2);

{ID}            return length_check(T_IDENTIFIER, MAX_LEN);   // 该规则写在保留词规则之后
```

词法分析：回顾

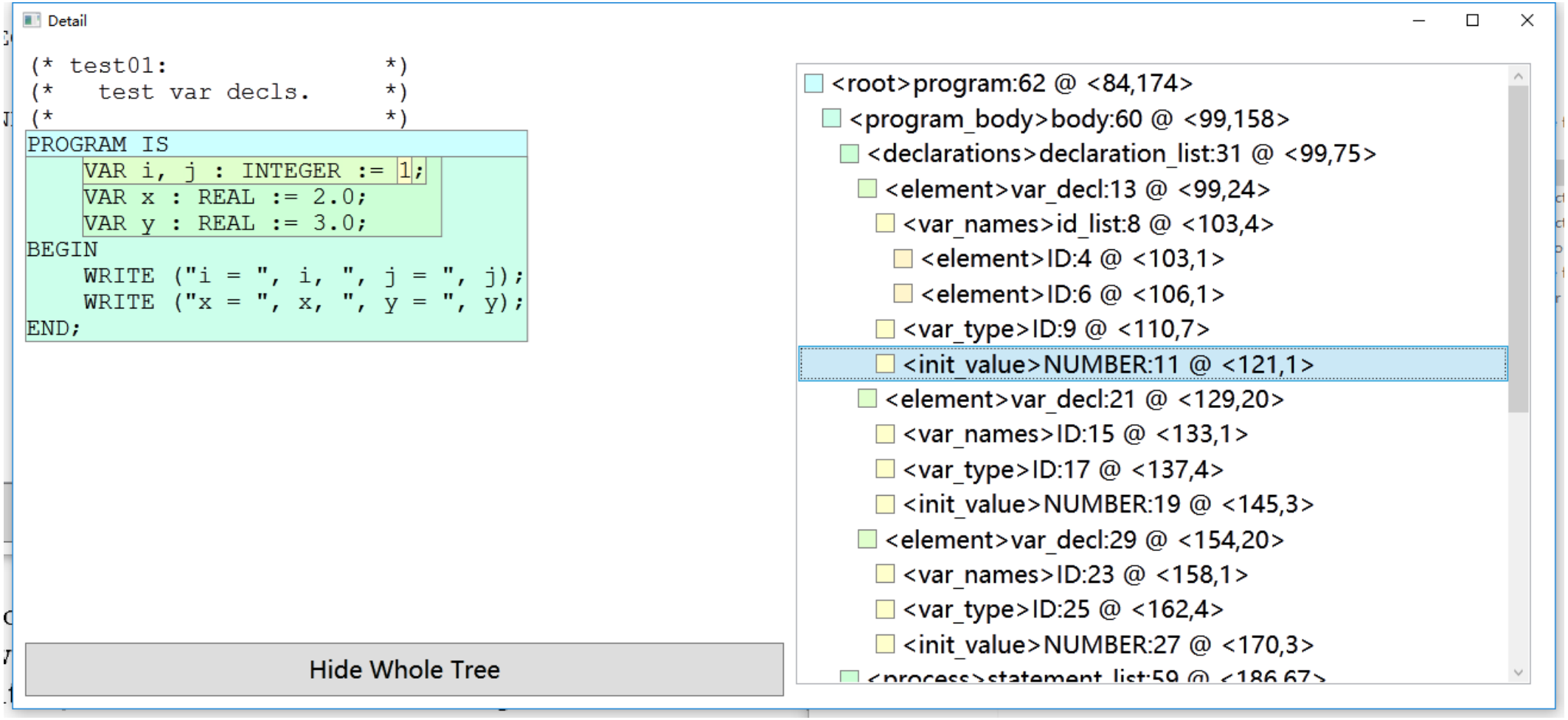
- 第三类：其他/统计行列信息

```
%{
static void do_before_each_action();
#define YY_USER_ACTION do_before_each_action();        // 定义每个识别操作之前的行为
int cur_line_num = 1, cur_col_num = 1;
}%
WS          [  \t]+

%%

{WS}                                                   // 遇到空格以及制表符，直接跳过
“(*”([^\)]|[^\*]\))*“*)“                               // 忽略注释信息
<<EOF>>                 return T_EOF;                  // 文本终结符返回结束信息
\n                      ++cur_line_num; cur_col_num = 1;   // 遇到回车，行数加一，列数重置

%%

static void do_before_each_action() {  cur_col_num += yyleng; }
```

词法分析：小结

- 1. yylex() 为调用的接口，在lex规则处理时没有遇到return语句，该函数不会停止，直到本文识别结束
- 2. 识别到的字符串会存储在变量char* yytext中
- 3. 识别到的字符串长度信息会存储在变量int yyleng中
- 4. 通过#define YY_USER_ACTION function，可以设置每个处理时的前置操作
- 5. 学会使用基本的正则表达式

句法分析


Detail
(* test01: *)
(* test var decls. *)
(* *)
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;
Hide Whole Tree
<root>program:62 @ <84,174>
<program_body>body:60 @ <99,158>
<declarations>declaration_list:31 @ <99,75>
<element>var_decl:13 @ <99,24>
<var_names>id_list:8 @ <103,4>
<element>ID:4 @ <103,1>
<element>ID:6 @ <106,1>
<var_type>ID:9 @ <110,7>
<init_value>NUMBER:11 @ <121,1>
<element>var_decl:21 @ <129,20>
<var_names>ID:15 @ <133,1>
<var_type>ID:17 @ <137,4>
<init_value>NUMBER:19 @ <145,3>
<element>var_decl:29 @ <154,20>
<var_names>ID:23 @ <158,1>
<var_type>ID:25 @ <162,4>
<init_value>NUMBER:27 @ <170,3>

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转换为C程序来解析该语法。
- .y文件的结构布局：

```
%{
C declarations
%}

Bison declarations

%%
Grammar rules
%%
Additional C code
```

实现一个简单的计算器

- Input: (2+3)*5+5
- Out: 30
- L1文法：

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

实现一个简单的计算器

- 分析文法中的token（终结符）
- Tokens:
  - +, -, *, /, (,), number
- 需要在Bison中定义token
- 识别token的部分在lexer进行

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

实现一个简单的计算器

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

- 编写过程： bison -> flex -> main
- demo.y:

```
%{
#include <iostream>
using namespace std;
#include "lex.c"
%}

%union {
   double val;  //定义yylval.val为double
}
%token <val> NUMBER
%token ADD SUB MUL DIV OP CP
%token EOL
%type <val> exp
%type <val> factor
%type <val> term
```

```
%%
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
- 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

- Demo: https://github.com/WangJY06/CompilerPJ/tree/master/PJ2_demo
- Bison参考资料: http://ranger.uta.edu/~fegaras/cse5317/bison/bison_toc.html
- PCAT语言参考中有相应的LL1文法参考