hw3_4实验过程记录

参考资料

基本功能参考hw1。

错误处理部分参考hw2/docs/reading/lexandyacc.pdf的第九章：Error Reporting and Recovery。

实验过程

Task 1：完善parser基本功能

1. Yacc：根据AST，定义yylval

- 直接将 `fdmjast.h` 中的节点类型照搬过来就可以
- 关于token `ID`：尽管大多情况下Yacc只需要其字符串，但在 `Exp` 的生成式中需要其position信息，故其 `yylval` 设为 `IdExp` 类型

2. Yacc：根据语法，定义termianl symbols（tokens）、non-termianl symbols、start symbol

- 直接根据给出的语法定义终结符和非终结符即可

3. Lex：根据词法，编写词法解析规则，在代码部分将token的值填充进yylval，并将token返回给Yacc

- 状态机：
  - 状态的定义和hw1完全类似：`INITIAL`，`COMMENT1`，`COMMENT2`
  - 注释的解析规则完全参照hw1
- 解析token
  - 直接匹配并解析即可
  - 可以分别匹配整型 `integer` 和浮点型 `float`，都返回token `NUM`
- 注意需要将匹配 `ID` 的规则放在所有匹配关键字的规则之后
- 注意相较hw1，`!` 变成一元求反运算符，需要从正则表达式别名 `punctuation` 中去掉

4. Yacc：根据语法，定义precedence，编写语法解析规则，在代码部分生成AST

- 规则部分照搬FDMJ.md中的语法即可
- 考虑运算符优先级问题 (从低到高)
  - 逻辑或 `||`：左结合
  - 逻辑与 `&&`：左结合
  - 相等 `==`，不等 `!=`：左结合

- 小于 `<`，大于 `>`，小于等于 `<=`，大于等于 `>=`：左结合
- 加 `+`，减 `-`：左结合
- 乘 `*`，除 `/`：左结合
- 负号 `-`，逻辑非 `!`：右结合
- 对象成员 `.`，数组下标 `[]`，函数调用 `()`：左结合，其中在Yacc中定义precedence时只需考虑左中括号和左小括号

其中，由于二元运算符减号和一元运算符负号 `-` 的token相同，增加一个优先级和 `!` 相同的伪token `UMINUS`，在相应规则的尾部添加 `%prec UMINUS`，从而将 `MINUS` 的优先级提升到和 `UMINUS` 相等。

- 解决danling else问题：非运算符的优先级
  - 当yacc的格局为如下所示时

```
Stm -> IF ( Exp ) Stm . ELSE Stm
Stm -> IF ( Exp ) Stm .
```

会产生冲突：不知道是继续移进 `ELSE` 还是按下面的规则直接规约

  - 采取的原则：else应该和最近的if配对，即应该优先移进而非规约
  - 具体实现：

```
%precedence THEN
%precedence ELSE

STM: IF '(' EXP ')' STM ELSE STM {
  $$ = A_IfStm($1, $3, $5, $7);
} | IF '(' EXP ')' STM %prec THEN {
  $$ = A_IfStm($1, $3, $5, NULL);
} ...
```

让else所在的规则拥有更高的优先级，所以分析器会优先移进else而非直接规约。

Task 2：语法错误处理

- 主要思想：对错误的地方，直接跳过这一行，继续处理下面
- 解决的错误处理：
  1. 变量声明：若某一个VarDecl错误 (要求不能缺失开头的 `int / float / class` 以及末尾的 `;`)，可以跳过该变量声明继续处理余下的VarDeclList
  2. 语句错误：若某一个Stm错误 (要求不能缺失末尾的 `;` 或 `}`)，可以跳过该语句继续处理余下的StmList
  3. 类声明错误：若某个Class在声明时就已经出现错误，可以跳过该类继续处理余下的类
  4. 类中的方法声明错误：若某个Method在声明时就已经出现错误，可以跳过该方法继续处理余下的方法
- 方法：在VarDecl / Stm / ClassDecl / MethodDecl 的产生式中添加如下规则

- 以VarDecl为例

```
// example
VarDecl: ...
  | INT error ';' {
  yyerrok;
  $$ = NULL;
} | FLOAT error ';' {
  yyerrok;
  $$ = NULL;
} | CLASS error ';' {
  yyerrok;
  $$ = NULL;
};

VAR_DECL_LIST: /* empty */ {
  $$ = NULL;
} | VAR_DECL VAR_DECL_LIST {
  if ($1 != NULL) {
    $$ = A_VarDeclList($1, $2);
  } else {
    $$ = $2;
  }
};
```

  这样遇到错误的变量声明 (开头的类型和末尾的 `;` 不能缺失，否则无法处理) 就可以跳过这一行，继续处理剩余的变量声明。

- `yyerrok` 的作用是让处理器及时从error recovering模式中恢复到正常模式，若有紧挨着的error可以避免不输出错误信息

- 输出错误信息：`yyerror` 函数

```
void yyerror(char *s) {
  extern int pos, line;
  extern char *yytext;
  extern int yyleng;
  extern char linebuf[2000];
  fprintf(stderr, "line %d,%d: %s near %s:\n%s\n", line, pos - yyleng, s,
yytext, linebuf);
  fprintf(stderr, "%*s\n", pos - yyleng, "^");
}
```

  - 完全参考 lexandyacc.pdf P237 (255/354) 的错误报告写法
  - 可以打印出错的那一行，并用 `^` 指向开始发生错误的地方

测试结果

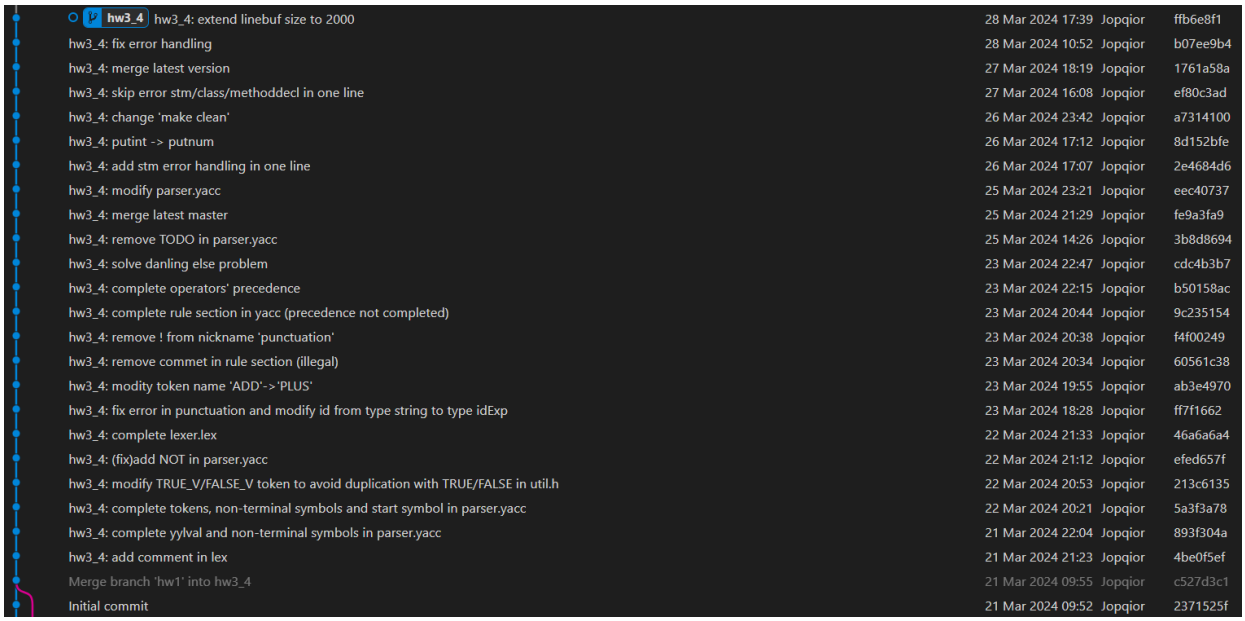
使用提供的fmj2ast生成.src和.ast后，重命名为.debug.src和.debug.ast，然后移动到hw3_4/test下，运行python测试脚本：

```
zqwh@LAPTOP-HDCBVNK7:~/compiler/2024/hw3_4/test$ python3 ast_src_diff.py
test1: True
intbubblesort: True
bubblesort: True
example: True
float: True
fibonacci: True
zqwh@LAPTOP-HDCBVNK7:~/compiler/2024/hw3_4/test$ python3 ast_xml_diff.py
test1: True
intbubblesort: True
bubblesort: True
example: True
float: True
fibonacci: True
```

测试全部通过。

开发过程

git提交记录如下：



开发流程基本按照上面叙述的实验过程进行。