

**杭州电子科技大学**

**《编译原理课程实践》**

**实验报告**

题 目： 1.词法分析程序

2.语法分析部分

3.语义分析部分

学 院： 卓越学院

专 业： 计算机英才班

班 级： 22187111

学 号： 22270336

姓 名： 成家康

完成日期： 2024/6/20

目录

1. 实验目的
2. 实验内容与实验要求
3. 设计方案和算法描述
4. 测试结果
5. 源代码

## 实验目的

### 实验⼀ 词法分析程序

利用 flex 工具生成SysY2022语言的词法分析器，要求输入一个SysY2022语言源程序文件,比如test.c，词法分析器能输出该程序的token以及token的种别。

### 实验二 语法分析程序

利用flex+bison生成SysY2022的语法分析程序。要求任给一个SysY2022语言的源程序，输出其抽象语法树(AST)。

### 实验三 语义分析程序

完成SysY2022的语法分析程序。要求任给一个SysY2022语言的源程序，能识别源程序中出现的语义错误。

## 实验内容与实验要求

### 实验⼀ 词法分析程序

实验内容

实验⼀的任务是编写⼀个程序，对使⽤SysY语⾔书写的源代码进⾏词法分析（SysY语⾔的⽂法参考附件⽂件,2022年版本的SysY语⾔增加了float基本数据类型，其他基本上⼀致，float部分作为选做内容），并打印分析结果。实验的实现⽅式可以⼿⼯编写程序，也可以采⽤词法分析⼯具GNU Flex等。

需要注意的是，由于在后⾯的实验中还会⽤到本次实验已完成的代码，因此，建议保持良好的代码⻛格，系统地设计代码结构和各模块之间的接⼝。

实验要求

程序要能够查出SysY源代码中可能包含的词法错误:

词法错误(错误类型代码为A)：出现SysY词法中未定义的字符以及任何不符合SysY词法单元定义的字符。

程序在输出错误提示信息时，需要输出具体的错误类型、出错的位置（源程序的⾏号）以及相关的说明⽂字。

### 实验二 语法分析程序

实验内容

实验⼆的任务是在实验⼀的词法分析程序基础上，编写⼀个程序，对使⽤SysY语⾔书写的源代码进⾏语法分析，并打印分析结果。实验的实现⽅式可以⼿⼯编写程序，也可以采⽤词法分析⼯具GNU Flex和语法分析⼯具 Bison等。

需要注意的是，由于在后⾯的实验中还会⽤到本次实验已完成的代码，因此，建议保持良好的代码⻛格，系统地设计代码结构和各模块之间的接⼝。

实验要求

程序要能够查出SysY源代码中可能包含的词法错误和语法错误:

词法错误(错误类型代码为A)：

出现SysY词法中未定义的字符以及任何不符合SysY词法单元定义的字符。

语法错误(错误类型代码为B).

程序在输出错误提示信息时，需要输出具体的错误类型、出错的位置（源程序的⾏号）以及相关的说明⽂字。

### 实验三 语义分析程序

实验内容

实验三的任务是在词法和语法分析的基础上，编写⼀个程序，对使⽤SysY语⾔书写的源代码进⾏静态语义检查，并打印分析结果。实验的实现⽅式可以⼿⼯编写程序，也可以探索在Bison中增加语义检查规则来实现。

需要注意的是，由于在后⾯的实验中还会⽤到本次实验已完成的代码，因此，建议保持良好的代码⻛格，系统地设计代码结构和各模块之间的接⼝。

实验要求

程序要能够查出SysY源代码中可能包含的语义错误。本次实验中，SysY语⾔的语义特性符合SysY语⾔定义(2022版)中的语义约束描述，我们简单归纳如下：

1. 特性1：⼀个SysY程序由单个⽂件构成，⽂件内容对应语法单元为CompUnit，其中必须存在⼀个标识符为“main”、⽆参数、返回类型为int的FuncDef（函数定义）

2. 特性2：每个CompUnit对应的顶层变量或常量声明语句、函数定义，都不能重复定义同名标识符，即使类型不同也不⾏。【简单的说，就是不允许在同个作⽤域下具有同名的标识符（包括变量、常量、函数名）】

3. 特性3：任何函数只进⾏⼀次定义，⽆法进⾏函数声明【同学们可以关注⼀下SysY语⾔定义中的FuncDef】

4. 特性4：与C语⾔标准⼀致，函数⽆法进⾏嵌套定义，即函数内部不能再定义函数

5. 特性5：作为if和while条件部分的语法单位Cond是⼀个LOrExp，需要特别注意逻辑运算表达式，单⽬运算符！只能出现在LOrExp，⽽不能在Exp中出现。

6. 特性6：函数调⽤时实际参数和函数定义中的参数个数和类型必须保持完全匹配。

7. 特性7：SysY允许类型隐式转换，本次实验我们假设int和float不能互相赋值或互相运算。

各⼩组的程序需要对输⼊⽂件进⾏语义分析（输⼊⽂件中可能包含函数、⼀维或多维数组），并检查如下类型的

错误：

1. 错误类型1：变量未声明

2. 错误类型2：变量重复声明

3. 错误类型3：函数在调⽤时未定义

4. 错误类型4：函数重复定义（同样的函数名出现了不⽌⼀次定义）

5. 错误类型5：把变量当做函数调⽤，如对普通变量使⽤括号(...)或()运算符（当函数调⽤），

6. 错误类型6：对函数名的不当引⽤（如把函数名当做普通变量来引⽤）

7. 错误类型7：对数组的不当引⽤，如数组访问运算符"[...]"中出现⾮整数表达式，即数组变量的下标不是整型

8. 错误类型8：对⾮数组变量使⽤数组访问"[...]"运算符

9. 错误类型9：函数调⽤时参数个数或类型不匹配

10. 错误类型10：return语句返回的类型与函数定义的返回类型不匹配

11. 错误类型11：操作数类型不匹配，或操作数类型与操作符不匹配，如：整型变量与数组变量相加减，或数组变量与数组变量相加减。

12. 错误类型12：break语句不在循环体内

13. 错误类型13：continue语句不在循环体内

注意：语法分析能发现的错误不包含在上⾯要求⾄少能识别上述3个以上的错误类型，程序在输出错误提示信息时，需要输出具体的错误类型、出错的位置（源程序的⾏号）以及相关的说明⽂字。

## 设计方案与算法描述

### 实验⼀ 词法分析程序

一开始是通过自己上网查阅资料去设计了一个sysy的词法分析器，后面用头歌上的相关资源去设计了一个sysy的词法分析器，下面以第二个词法分析器为例，设计方案主要是通过下图的正则表达式设计来匹配各个语句，开始用. {printf("Lexical error - line %d : %s\n",yylineno,yytext);return LEX\_ERR;}来除去所有sysy未定义的字符，然后分别匹配下面正则表达式来输出结果：

NONZERO: [1-9]  
匹配非零数字（1到9）。

DIGIT: [0-9]  
匹配任何数字（0到9）。

LETTER: [A-Za-z]  
匹配任何字母（大小写）。

OCTAL\_DIGIT: [0-7]  
匹配八进制数字（0到7）。

OCTAL\_CONST: 0{OCTAL\_DIGIT}\*  
匹配八进制常量，以0开头，后跟零个或多个八进制数字。

ILLEGAL\_OCTAL\_CONST: 0[0-9a-wy-zA-WY-Z]({LETTER}|{DIGIT})\*  
匹配非法的八进制常量，以0开头，后跟非法字符。

HEX\_PREFIX: 0x|0X  
匹配十六进制前缀。

HEX\_DIGIT: [0-9a-fA-F]  
匹配十六进制数字。

HEX\_CONST: {HEX\_PREFIX}{HEX\_DIGIT}+  
匹配十六进制常量，以0x或0X开头，后跟一个或多个十六进制数字。

ILLEGAL\_HEX\_CONST: {HEX\_PREFIX}({LETTER}|{DIGIT})\*  
匹配非法的十六进制常量。

NONDIGIT: {LETTER}|"\_"  
匹配非数字字符或下划线。

ID: {NONDIGIT}({DIGIT}|{NONDIGIT})\*  
匹配标识符，以非数字字符或下划线开头，后跟零个或多个数字或非数字字符。

DEC\_CONST: {NONZERO}{DIGIT}\*  
匹配十进制常量，以非零数字开头，后跟零个或多个数字。

ILLEGAL\_DEC\_CONST: {DIGIT}+{LETTER}+{DIGIT}+  
匹配非法的十进制常量，由数字、字母和数字组成。

FLOAT\_CONST: ({DEC\_CONST}?"."{DIGIT}\*)|(({DEC\_CONST}|{OCTAL\_CONST}|{HEX\_CONST})[eE][-+]?{HEX\_DIGIT}\*)  
匹配浮点常量，可以是十进制、小数或科学计数法形式。

ILLEGAL\_FLOAT\_CONST: {DIGIT}+{LETTER}+  
匹配非法的浮点常量，由数字和字母组成。

MultilineComment: "/\*"([^\\*]|(\\*)\*[^\\*/])\*(\\*)\*"\*/"  
匹配多行注释，以/开头，后跟任意字符，最后以/结尾。

SingleLineComment: "//".\*$  
匹配单行注释，以//开头，后跟任意字符，直到行尾。

### 实验二 语法分析程序

语法分析器是根据头哥上的资料设计的，一开始没有打开c++版本所以缺失相关的makefile以及展示的printer.h和printer.cpp文件，于是自己手写了一个相关的display来进行语法树的输出，主要就是先将语句产生式分解成所有能够产生的语句，在将相关的符号替换成宏定义，最后再调用newnode()来实现语法树结点的创建

Stmt : LVal ASSIGN Exp SEMICOLON {

            $$ = new\_node(Stmt,$1,NULL,$3,AssignStmt,0,NULL,NonType); }

        | Exp SEMICOLON {

            $$ = new\_node(Stmt,NULL,NULL,$1,ExpStmt,0,NULL,NonType); }

        | SEMICOLON {

            $$ = new\_node(Stmt,NULL,NULL,$1,BlankStmt,0,NULL,NonType); }

        | Block {

            $$ = new\_node(Stmt,NULL,NULL,$1,Block,0,NULL,NonType); }

        | IF LP Cond RP Stmt ELSE Stmt {

            $$ = new\_node(Stmt,$3,$5,$7,IfElseStmt,0,NULL,NonType);  }

        | IF LP Cond RP Stmt %prec THEN {

            $$ = new\_node(Stmt,$3,NULL,$5,IfStmt,0,NULL,NonType); }

        | WHILE LP Cond RP Stmt {

            $$ = new\_node(Stmt,$3,NULL,$5,WhileStmt,0,NULL,NonType); }

        | BREAK SEMICOLON  {

          $$ = new\_node(Stmt,NULL,NULL,NULL,BreakStmt,0,NULL,NonType); }

        | CONTINUE SEMICOLON {

       $$ = new\_node(Stmt,NULL,NULL,NULL,ContinueStmt,0,NULL,NonType); }

        | RETURN Exp SEMICOLON {

            $$ = new\_node(Stmt,NULL,NULL,$2,ReturnStmt,0,NULL,NonType); }

        | RETURN SEMICOLON {

  $$ = new\_node(Stmt,NULL,NULL,NULL,BlankReturnStmt,0,NULL,NonType); }

        ;

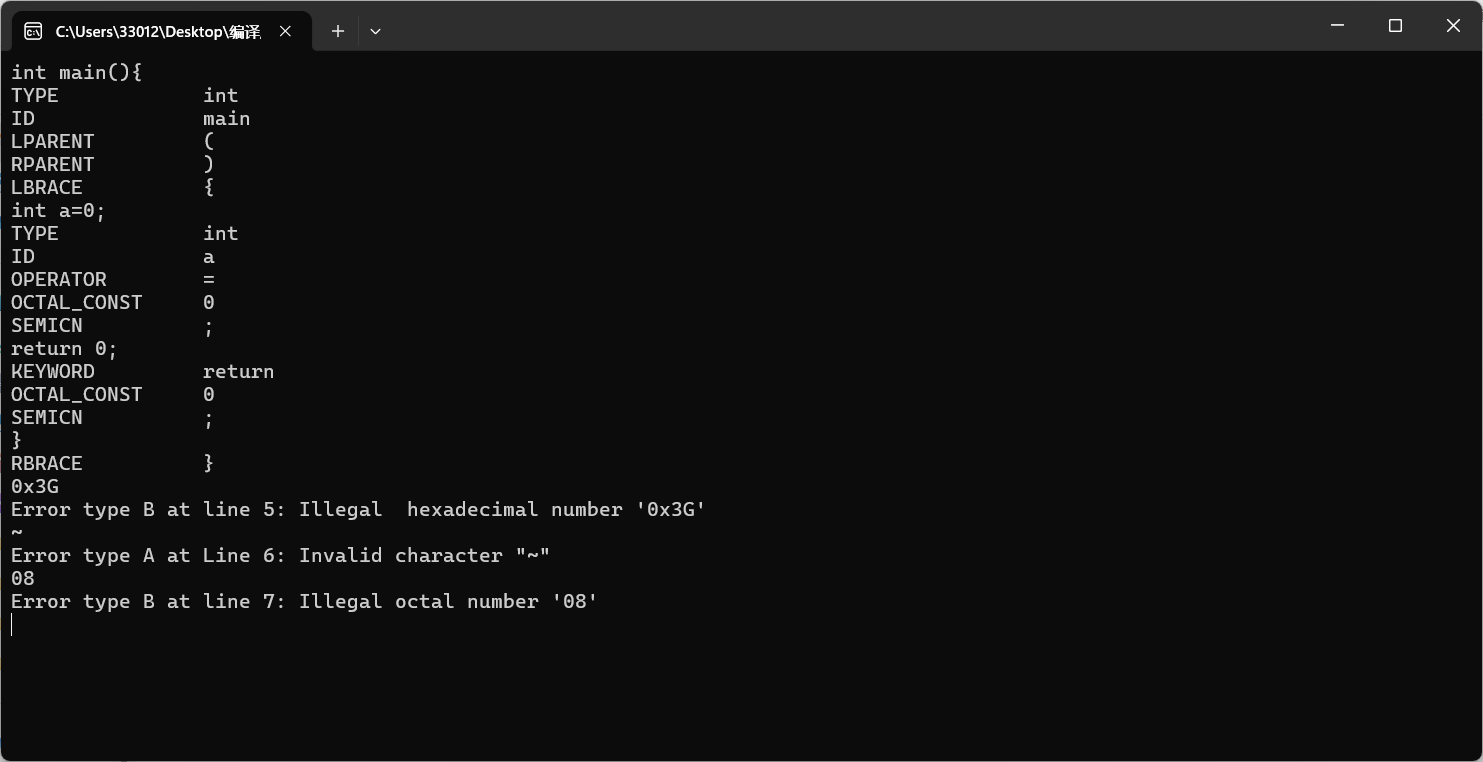
### 实验三 语义分析程序

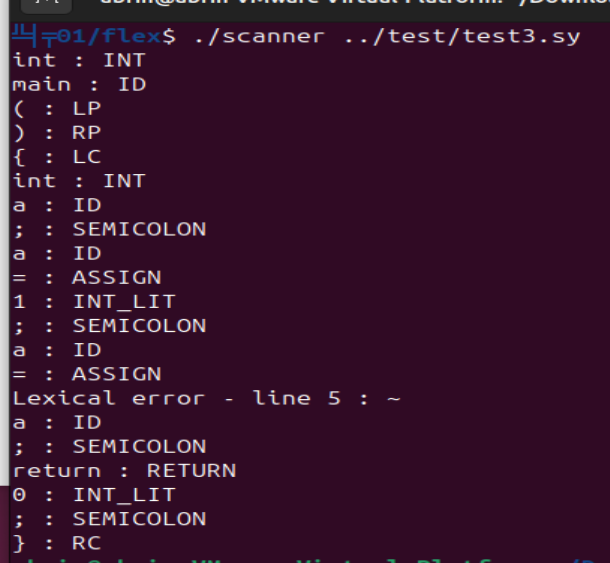
语义分析这一块因为c语言版本缺少了相关文件难度较大，故选择通过c++版本的语法分析来实现，通过对checker函数的阅读和修改完成头歌上的前11个测试点，然后本地部署时通过修改main文件来实现词法分析，语法分析，语义检查的连接

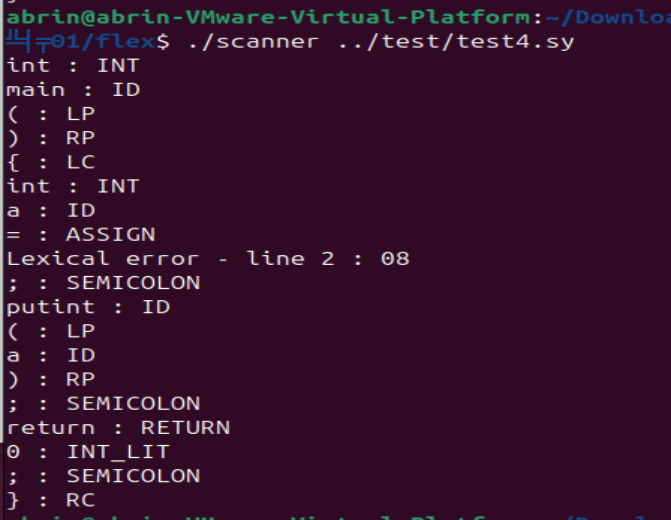
## 测试结果

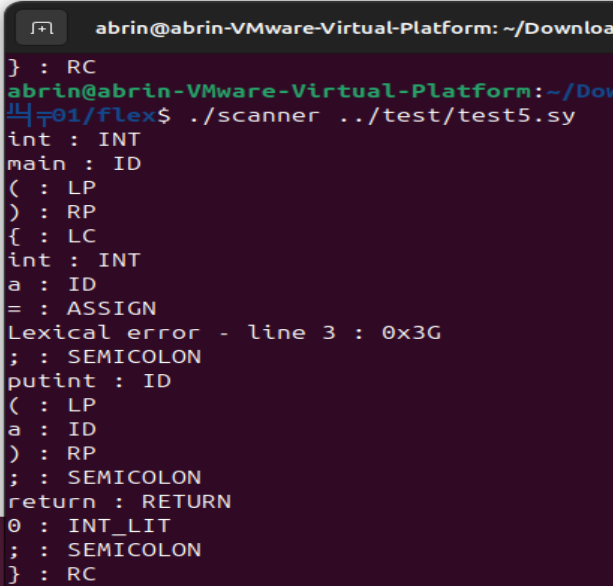
### 实验⼀ 词法分析程序

第一个图是自己设计的词法分析器，后面在虚拟机上的是匹配sysy程序的词法分析器

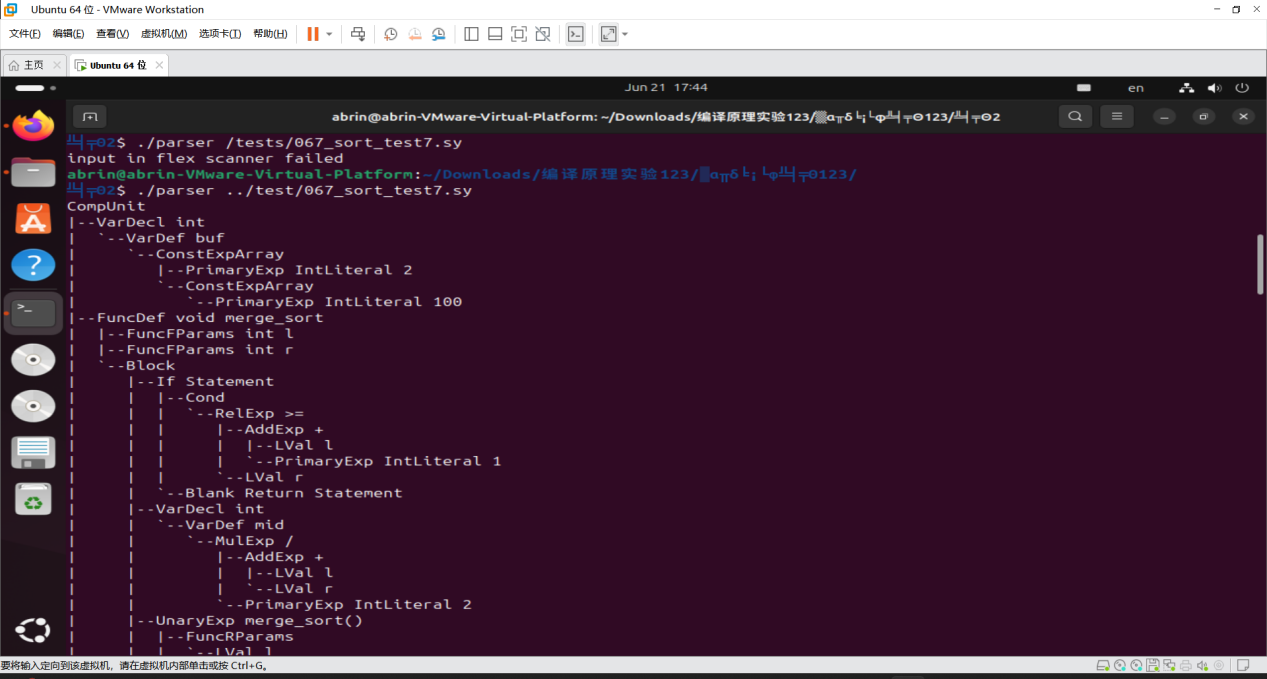


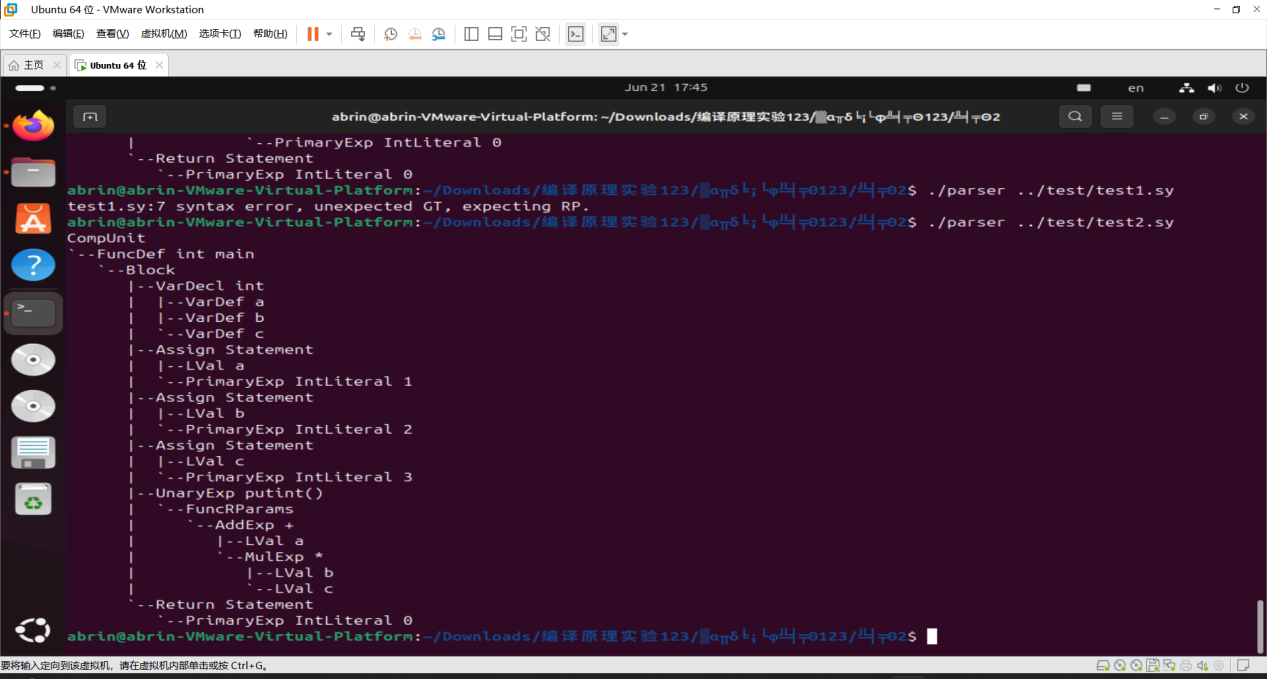
****

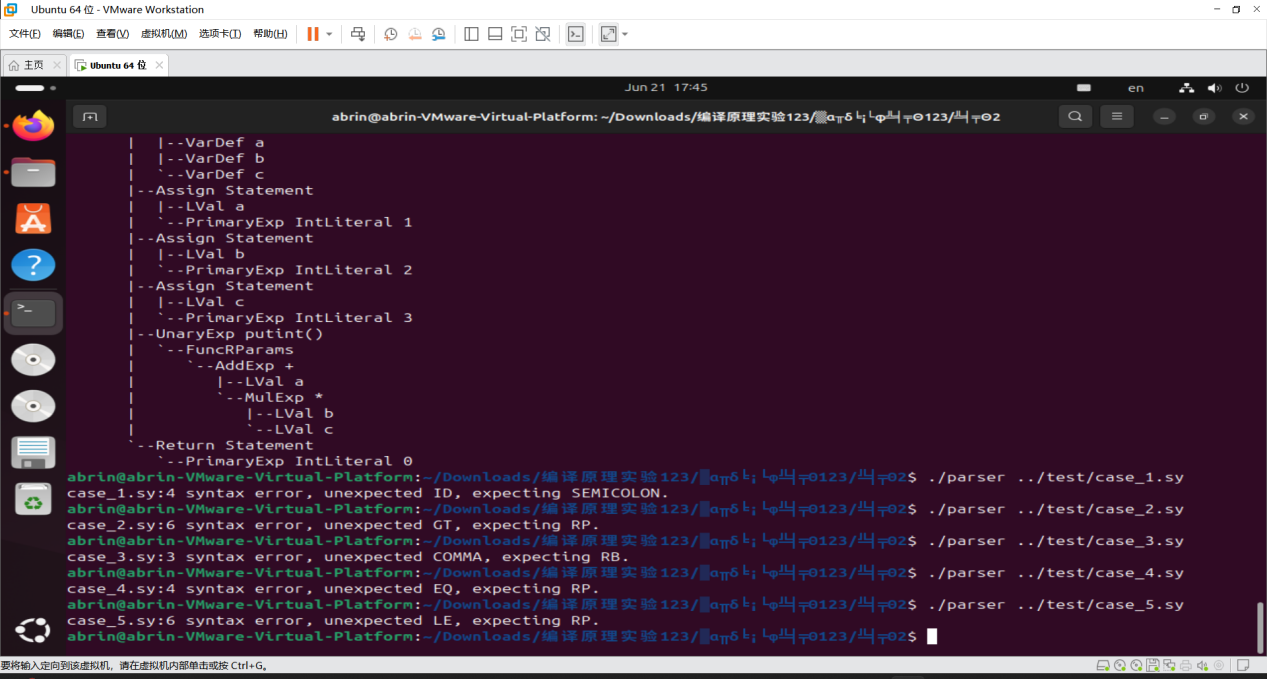
****

****

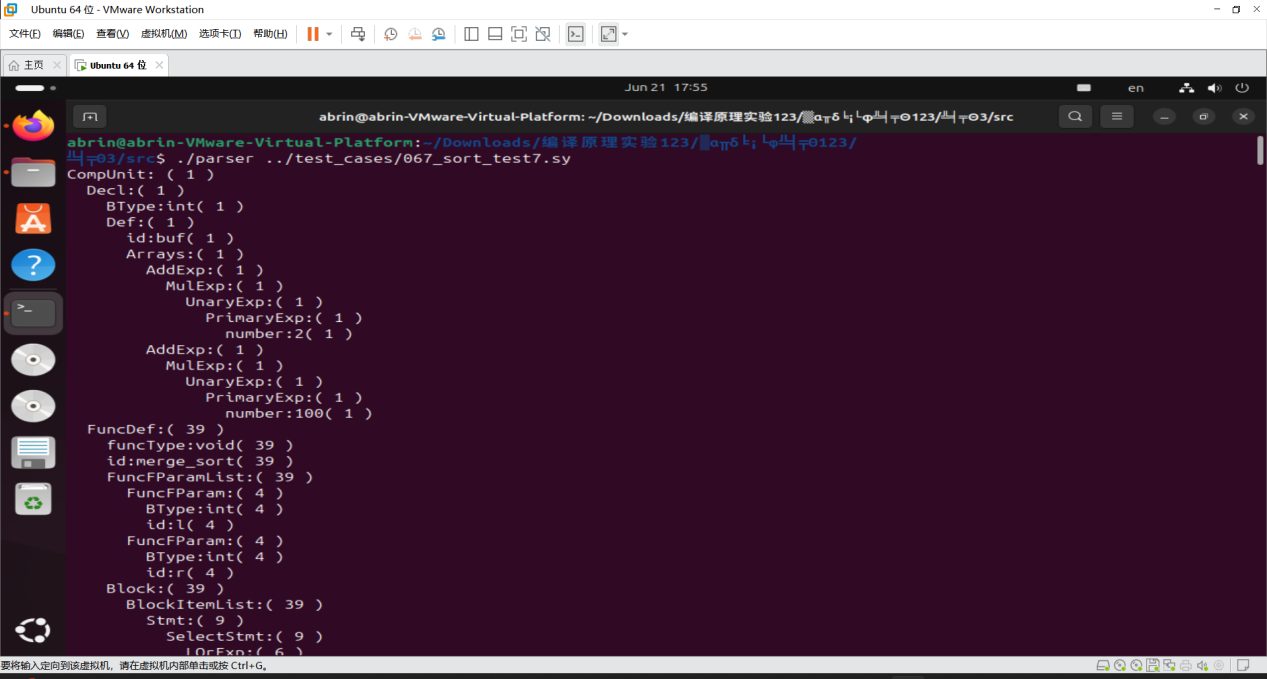
### 实验二 语法分析程序

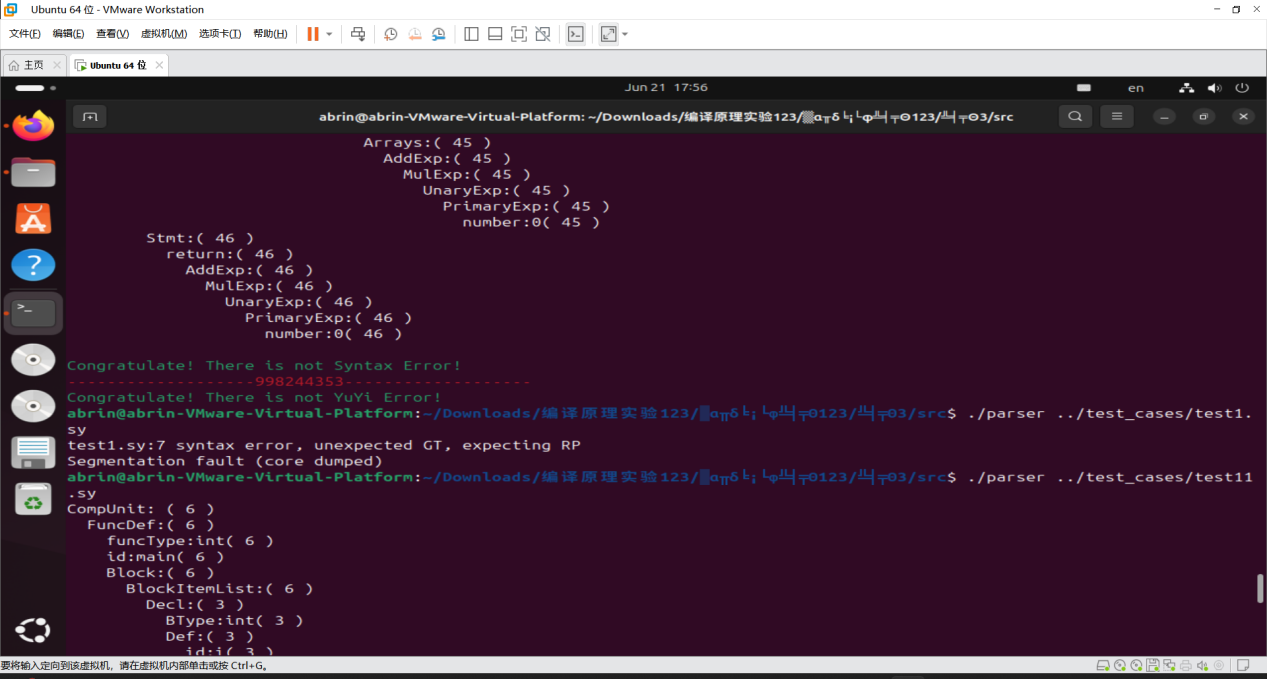


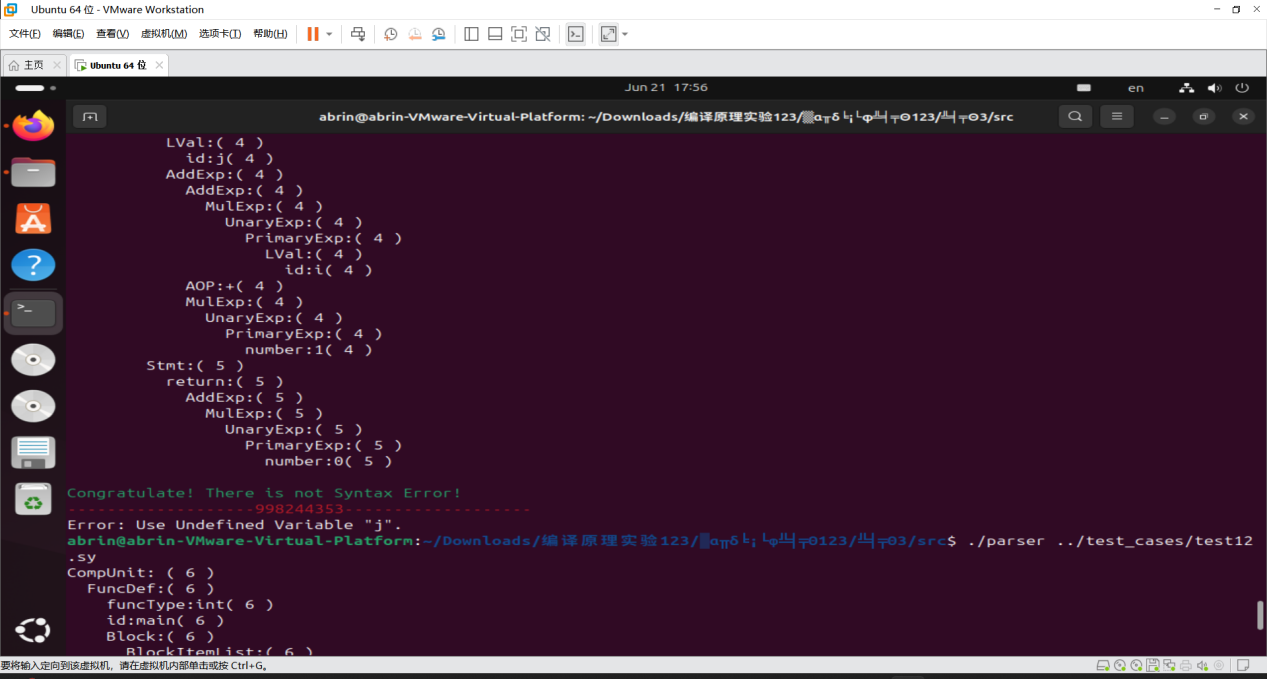
****

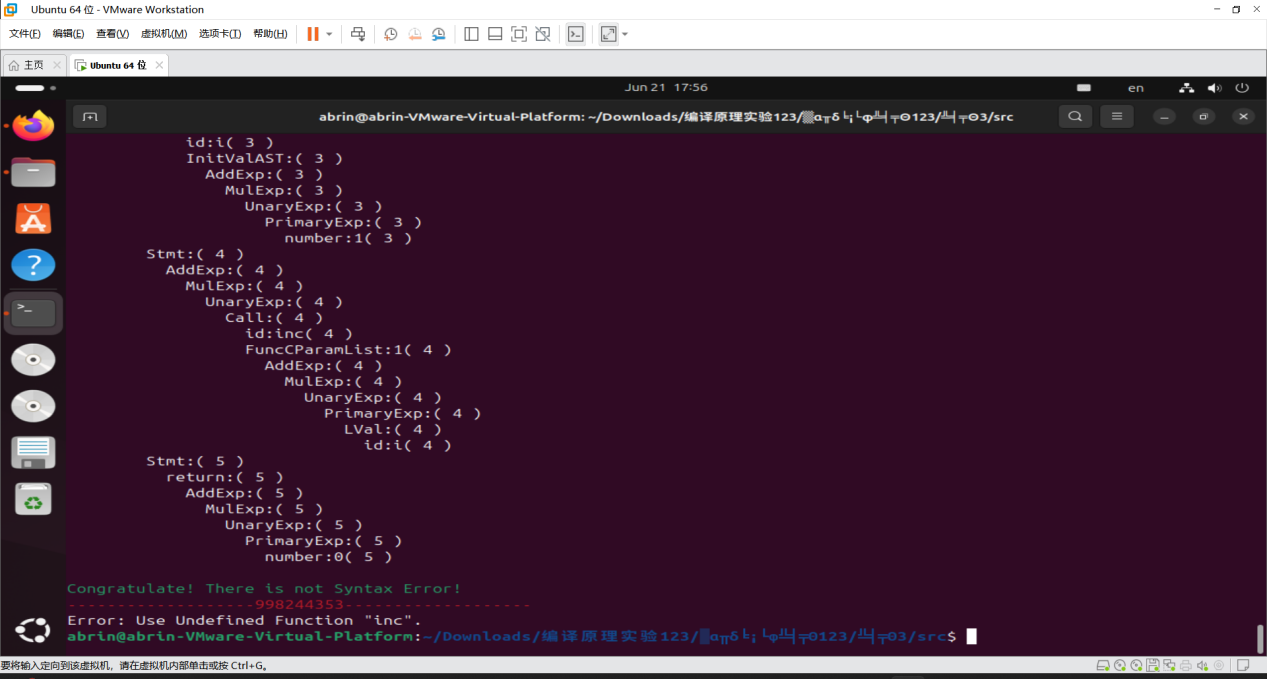
****

### 实验三 语义分析程序



****

****

****

## 源代码

