编译原理 实验1报告

实验人：顾馨兰 202220005　周家琛 202220019
邮箱 ： gu_xinlan@163.com
任务号：7
实验时间：2021年3月

一、实验内容

必做要求：对C－语言源代码进行词法分析和语法分析，识别程序中的词法或语法错误，并为正确的C－程序输出语法树。
选做要求1.1：识别八进制数和十六进制数。

二、运行方式

按照实验指定环境，在`Code/`目录下执行`make test`.

三、实验结果

1.语法树生成

按照实验要求完成，示例略。

2.词法/语法错误识别

ERROR TYPE A

识别所有不符合C--词法要求的符号：

```
int j = ~i;
```

```
ERROR TYPE A at line 1: Undefined character: ~
```

八进制和十六进制数字错误识别为词法错误：

```
int main(){
      int i = 09;
      int j = 0x3G;
}
```

```
ERROR TYPE A at line 3: Illegal octal number: 09
ERROR TYPE A at line 4: Illegal hexadecimal number: 0x3G
```

ERROR TYPE B

部分能够识别并给出描述的语法错误示例如下：

```
int test(int x,int ){}
int test(,int y){}
int test()}
a;
int a                    //被识别为下一行的错误
int a = 5;
int main(){
      int a,;
      a[1][1 = 1;
      a[5,3] = 1.5;      //被识别为4次错误
      test(a,);
      test(;
      test);
      ;
      j = a+;
      if () a; else b;
      if (x) a else b;
      while (int) b;
      return int;

```

```
ERROR TYPE B at line 1: Expected VarDec.
ERROR TYPE B at line 2: Expected ParamDec.
ERROR TYPE B at line 3: Expected "{".
ERROR TYPE B at line 4: Expected Specifier.
ERROR TYPE B at line 6: Expected ";" after Def.
ERROR TYPE B at line 6: Unexpected token following VarDec.
ERROR TYPE B at line 8: Expected DecList.
ERROR TYPE B at line 9: Expected  "]".
ERROR TYPE B at line 10: Expected  "]".
ERROR TYPE B at line 10: Expected ";" after Stmt.
ERROR TYPE B at line 10: Expected ";" after Stmt.
ERROR TYPE B at line 10: Incorrect Stmt.
ERROR TYPE B at line 11: Expected Args between "()".
ERROR TYPE B at line 12: Expected Args or ")".
ERROR TYPE B at line 13: Expected ";" after Stmt.
ERROR TYPE B at line 13: Incorrect Stmt.
ERROR TYPE B at line 14: Incorrect Stmt.
ERROR TYPE B at line 15: Expected Exp.
ERROR TYPE B at line 16: Expected Exp in "()".
ERROR TYPE B at line 17: Expected ";" after Stmt.
ERROR TYPE B at line 18: Expected Exp in "()".
ERROR TYPE B at line 19: Expected Exp to return.
ERROR TYPE B at line 20: Expected "}".
```

四、实验思路

1.文法错误识别：遇到的主要困难在于解决在文法中增加error产生式后出现的冲突，在实验过程中发现有些冲突较容易通过设置优先级解决，有些冲突很难找到理想的解决方案。例如对于上方代码末尾缺失`}`的情况，一开始按下方第2行编写了识别缺失`}`符号的error产生式用于识别，但与第5行的错误产生式存在移入/规约冲突，分析器优先选择移入，所以读至程序末尾时分析器仍在期待`;`以按第5行的产生式规约，而不会按照第2行的产生式规约，导致无法识别最后缺失的`}`。

```
CompSt: LC DefList StmtList RC
      | LC DefList StmtList error   {myerr("Expected \"}\".");}
      ...
Stmt: Exp SEMI
      | error   SEMI            {myerr("Incorrect Stmt.");}
      ...
```

解决方案：由于第5行的错误产生式非常关键，不可缺少，因此考虑调整第2行的错误产生式写法。去掉了产生式中的`error`，将缺失`}`的`CompSt`设置为优先级最低的规约方案。

```
CompSt: LC DefList StmtList RC
      | LC DefList StmtList %prec LOWER_THAN_ANYTHING   {myerr("Expected \"}\".");}
      ...
```

通过确认`syntax.output`文件，发现这样写在如下状态会导致多个移入/规约冲突，但对于所有属于 $FIRST(Stmt)$ 的符号，都会选择移入，之后按照 $StmtList \rightarrow Stmt\ StmtList$ 规约，而对于其他输入符号会立刻按照 $StmtList \rightarrow \epsilon$ 规约。这正符合文法要求，因此对这些冲突不另做处理。

```
State 41
   35 CompSt: LC DefList . StmtList RC
   37       | LC DefList . StmtList
    ...
```

对于缺失`;`的`ExtDef`也采用相同方法处理。

2.八进制与十六进制数处理

我们认为八进制和十六进制数字错误应该识别为语法错误，一开始没有在词法部分进行处理，而留到语法分析中报错。然而在实际测试中，因为上述语句出现在`{}`中，`9;`将被识别为一句`Stmt`，而随后第4行的语句为`Def`，根据文法

$$CompSt \rightarrow LC\ DefList\ StmtList\ RC$$

`Stmt`后不能再出现`Def`，又因为实现了识别缺失`}`错误的功能，因此程序认为应在第4行前插入`}`，并将下一条语句作为`ExtDef`处理。而文法又不允许在外部变量定义中进行赋值操作，因此会再报一次错误。相当于同一行存在超过一个文法错误，导致报错结果并不理想。

```
int main(){
      int i = 09;
      int j = 0x3G;
}
```

```
ERROR TYPE B at line 3: Expected ";" after definition.
ERROR TYPE B at line 4: Expected "}".
ERROR TYPE B at line 4: Unexpected token following VarDec.
```

所以改为将八进制与十六进制数字错误按词法错误处理。

五、实验分工

- 顾馨兰：编写含错误产生式的语法代码（syntax.y），编写错误样例进行测试，编写实验报告；
- 周家琛：编写词法分析代码（lexical.l）、完成语法树的构建（syntax_tree模块）。