编译原理 Lab1

202220013 徐简

项目环境:

1. GNU LINUX Realease: Ubuntu
2. GCC version: 7.5.0
3. GNU Flex version:2.6.4
4. GNU Bison version: 3.0.4

实现的功能

词法分析器

- 根据附录中的C--语言文法，定义相应的词法单元
- 对部分不合法的词法单元进行处理，打印相应的错误信息，主要是通过预测可能的错误实现（浮点数，十六进制，八进制数，整数错误）然后书写相应的正则表达式实现，示例如下：

```
int main()
{
  int i = 1;
  int j = 2;
  int 6_Wrong;
  int x=0x12FZ;
}
```

```
Error type A at Line 5: Illegal ID '6_Wrong'.
Error type A at Line 6: Illegal hexadecimal int '0x12FZ'.
```

- 按照实验手册中推荐的方法，记录词法单元的位置信息

```
/* record appear location*/
int yycolumn = 1;
#define YY_USER_ACTION \
    yylloc.first_line = yylloc.last_line = yylineno; \
    yylloc.first_column = yycolumn; \
    yylloc.last_column = yycolumn + yyleng - 1; \
    yycolumn += yyleng;
```

语法分析器

- 首先实现语法分析的树形结构--一个多叉树，并实现建树和遍历的相关函数。flex里为构建终结符节点，bison相应的位置构建非终结符节点，并将终结符节点连接到相应的非终结符节点上，实现了多叉树的建立

```
// tree node n-ary tree
typedef struct {
```

```
    char* name;
    MyType type;
    int line;
    union {
        unsigned type_int;
        float type_float;
        char type_str[40];
    } val;

    struct Node* child;
    struct Node* next;
} Node;
```

- 通过bison实现语法分析器，使用手册方法，进行二义性与冲突处理

```
%right <node> ASSIGNOP
%left <node> OR
%left <node> AND
%left <node> RELOP
%left <node> PLUS MINUS
%left <node> STAR DIV
%right <node> NOT
%left <node> LP RP LB RB DOT
```

- 完成了一些语法分析错误恢复，通过yyerror，输出相应的报错信息

```
int main()
{
  int i = 1;
  int x =0xzz;
  if(i<1){
    i=1;


}
```

```
Error type B at Line 10: syntax error
```

测试

- 使用符合C--语言语法的文件作为parser输入，输出正常
- 主要测试方法是，生成具有各种类型错误的C--语言文件，并用完成的parser对其进行解析，输出为相应的错误信息

```
./parser ../Test/test1.cmm
Error type A at Line 4: Illegal hexadecimal int '0xzz'.
Error type B at Line 11: syntax error, unexpected TYPE.
Error type B at Line 13: Wrong expression or Definition after statement.
Error type B at Line 14: syntax error, unexpected RP.
Error type B at Line 14: Possibly missing ";" at this or last line.
Error type B at Line 14: syntax error, unexpected RP.
Error type B at Line 14: Wrong expression or Definition after statement.
Error type B at Line 20: syntax error, unexpected IF, expecting ID or SEMI.
Error type B at Line 20: Possibly missing ";" at this or last line.
Error type B at Line 20: syntax error, unexpected IF, expecting $end or STRUCT or TYPE.
Error type B at Line 22: Missing type or Wrong type.
Error type B at Line 23: syntax error, unexpected IF, expecting $end or STRUCT or TYPE.
Error type B at Line 24: Missing type or Wrong type.
Error type B at Line 25: syntax error, unexpected ELSE, expecting $end or STRUCT or TYPE.
Error type B at Line 26: Missing type or Wrong type.
Error type B at Line 27: syntax error, unexpected RC, expecting $end or STRUCT or TYPE.
Error type B at Line 30: Missing type or Wrong type.
Error type B at Line 31: syntax error, unexpected ID, expecting $end or STRUCT or TYPE.
Error type B at Line 31: Missing type or Wrong type.
Error type B at Line 32: syntax error, unexpected WHILE, expecting $end or STRUCT or TYPE
.
Error type B at Line 33: Missing type or Wrong type.
Error type B at Line 34: syntax error, unexpected ID, expecting $end or STRUCT or TYPE.
Error type B at Line 34: Missing type or Wrong type.
Error type B at Line 35: syntax error, unexpected RC, expecting $end or STRUCT or TYPE.
```

- 参考了该仓库中部分基础测试样例https://github.com/massimodong/compilers-tests

使用方法

在terminal中输入相应指令，完成编译、运行、测试：

- `flex lexical.l`
- `bison -d syntax.y`
- `gcc main.c syntax.tab.c -lfl -ly -o parser`
- `./parser mytest1.cmm`

实验思考

- 实验过程中，一定要先详细阅读理解C--语言的规则。由于没仔细阅读语言贵法，浪费了后续的大量精力
- 测试样例在lab1中十分重要，由于时间原因，并没有使用很强的测试样例，手动编写的一些文件，很难全面衡量parser的完善程度