编译原理实验一

zyc | 今天创建

程序编译与运行

在Makefile存在的目录下，执行下列命令。

```
make build
make rerun
```

之后可以运行测试用例：

```
make my_test num=1
make my_test num=2
...
make my_test num=10
```

程序实现的功能

基础功能

词法分析

包括C--语言关键字的识别，基本错误类型的报错。

语法分析

包括C--语言基本文法的识别，基本错误类型的报错。

个人较为特别的实现内容

词法分析

```
FLOAT [1-9]{DIGIT}*"."{DIGIT}+|0"."{DIGIT}+|{DIGIT}*"."{DIGIT}+[eE][+-]?{DIGIT}+

"."{DIGIT}+ {LexError=1;printf("Error type A at Line %d: Illegal floating point number \"%s\".\n", yylineno, yytext);}
{DIGIT}+"." {LexError=1;printf("Error type A at Line %d: Illegal floating point number \"%s\".\n", yylineno, yytext);}
{DIGIT}*"."{DIGIT}+[eE] {LexError=1;printf("Error type A at Line %d: Illegal floating point number \"%s\".\n", yylineno, yytext);}
{DIGIT}*"."{DIGIT}*[eE] {LexError=1;printf("Error type A at Line %d: Illegal floating point number \"%s\".\n", yylineno, yytext);}
```

浮点数的识别采用三种模式：

- 整数部分大于0的浮点数，不使用科学计数法。
- 整数部分为0的浮点数，不使用科学计数法。
- 使用科学计数法的浮点数。

浮点数错误识别：

- .23类似的没有整数部分的浮点数
- 23.类似的没有小数部分的浮点数
- 23.e或23.E类似的没有E的指数部分的浮点数
- 23.2e或23.2E类似的没有E的指数部分的浮点数

```
OCTAL 0[0-7]*
HEXNUM 0x[0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F]+|0X[0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f|A|B|C|D|E|F]+

{OCTAL}+[8-9]+ {LexError=1;printf("Error type A at Line %d: Illegal octal number \'%s\'.\n", yylineno, yytext);}
0X[0-9a-zA-Z]*[g-zG-Z]+[0-9a-zA-Z]*|0x[0-9a-zA-Z]*[g-zG-Z]+[0-9a-zA-Z]* {LexError=1;printf("Error type A at Line %d: Illegal hexadecimal number \'%s\'.\n", yylineno, yytext);}
```

8进制数和16进制数的识别。

```
IdNew [0-9_a-zA-Z ;=+]
IdNew2 [0-9_a-zA-Z \n\r\t\[\];),}\-{=\*(\<\+]
STRING {IdNew}*
STRING2 {IdNew2}*
COMMENT "//"
BLOCKCOMMENTSIGNBEGIN "/*"
BLOCKCOMMENTSIGNEND "*/"
LINECOMMENT {COMMENT}{WS}*{STRING2}*\n
BLOCKCOMMENT {BLOCKCOMMENTSIGNBEGIN}{WS}*{STRING2}*{BLOCKCOMMENTSIGNEND}
ERRORBLOCKCOMMENT {BLOCKCOMMENTSIGNBEGIN}{WS}*{STRING2}*
NESTBLOCKCOMMENT {BLOCKCOMMENTSIGNBEGIN}{WS}*{STRING2}*{BLOCKCOMMENT}{WS}*{STRING2}*{BLOCKCOMMENTSIGNEND}
```

行注释和块注释的识别：

行注释识别到 “**//**” 后，会一直忽略输入字符（在IdNew2中定义的），直至遇到“**\n**”。

块注释识别“**/***”和“***/**”，并一直忽略输入在这两个符号之间的输入字符。

块注释的嵌套识别：

块注释会就近识别匹配两个注释字符，所以如果在块注释中多加入了**/***或者***/**，就会导致可能无法正确按照用户的意图识别。

```
MIUASIGN "-="
INCADD "++"
POINTER "->"
STRUCTDOTOP "."
PERCENT "%"
```

可以识别一些额外的操作符。

语义分析

```
// Declarators
VarDec: ID
{$$=new_node(@$.first_line,NOTTOKEN,"VarDec",1,$1);}
    | VarDec LB INT RB
{$$=new_node(@$.first_line,NOTTOKEN,"VarDec",4,$1,$2,$3,$4);}
    | VarDec LB Exp RB
{$$=new_node(@$.first_line,NOTTOKEN,"VarDec",4,$1,$2,$3,$4);}
    | error INT                     {SynError=1;fprintf(stderr,"Error type B at line %d: Missing \"]\".\n", yylineno);}
    ;
```

二维数组错误识别：

只能识别当[]之间是整数表达式且定义出错的情况，若需要更多错误处理，可以在Exp表达式的各种情况来判断。

```
Stmt: Exp SEMI
{$$=new_node(@$.first_line,NOTTOKEN,"Stmt",2,$1,$2);}
    | Exp                           {SynError=1;fprintf(stderr,"Error type B at line %d: Missing \";\"\n", yylineno);}
```

```
    | CompSt
{$$=new_node(@$.first_line,NOTTOKEN,"Stmt",1,$1);}
    | RETURN Exp SEMI
{$$=new_node(@$.first_line,NOTTOKEN,"Stmt",3,$1,$2,$3);}
    | IF LP Exp RP Stmt
{$$=new_node(@$.first_line,NOTTOKEN,"Stmt",5,$1,$2,$3,$4,$5);}
    | IF LP Exp RP Stmt ELSE Stmt
{$$=new_node(@$.first_line,NOTTOKEN,"Stmt",7,$1,$2,$3,$4,$5,$6,$7);}
    | error SEMI                    {SynError=1;fprintf(stderr,"Error type B
at line %d: Missing \";\"\n", yylineno);}
    | WHILE LP Exp RP Stmt
{$$=new_node(@$.first_line,NOTTOKEN,"Stmt",5,$1,$2,$3,$4,$5);}
    | WHILE LP Exp SEMI RP Stmt     {SynError=1;fprintf(stderr,"Error type B
at line %d: redundant symbol \";\"\n", yylineno);}
    | STAR DIV                      {SynError=1;fprintf(stderr,"Error type B
at line %d: Syntax error\n", yylineno);}
    ;
```

Stmt错误分析：

Stmt是基本语句的组成，错误类型如下：

- 语句结束缺少“;”，进行报错。
- WHILE语句中缺少";"，报错。
- WHILE语句判断表达式结尾出现“;”，报错。
- ***/**为单独语句时，报错，这是为了迎合前面的块注释，当然，你也可以让它默认报错，不加这一行。

```
/*删除部分额外代码*/

ExtDefList: ExtDef ExtDefList
{$$=new_node(@$.first_line,NOTTOKEN,"ExtDefList",2,$1,$2);}
    |                               {$$=NULL;}
    ;

ExtDef: Specifier ExtDecList SEMI
{$$=new_node(@$.first_line,NOTTOKEN,"ExtDef",3,$1,$2,$3);}
    | Specifier SEMI
{$$=new_node(@$.first_line,NOTTOKEN,"ExtDef",2,$1,$2);}
    | Specifier FunDec CompSt
{$$=new_node(@$.first_line,NOTTOKEN,"ExtDef",3,$1,$2,$3);}
    ;
```

扩展ExtDef，使之可以是变量声明或者是函数声明。这样，就可以在函数声明之间穿插着变量声明。

```
FuncUse: ID LP Args RP
{$$=new_node(@$.first_line,NOTTOKEN,"FuncUse",4,$1,$2,$3,$4);}
    | ID LP RP
{$$=new_node(@$.first_line,NOTTOKEN,"FuncUse",4,$1,$2,$3);}
    | ID LP Args SEMI               {SynError=1;fprintf(stderr,"Error type B
at line %d: Missing \"\)\".\n", yylineno);}
    ;
```

函数调用的识别，这在书中的样例中没有提到

- 正常使用时funcname(params);或者 funcname();
- 若缺少"**)**"，则进行报错。

这部分在表达式中并没有加入，可以再进行扩展，实现如**int a = add(b,c);**这样的表达式的识别。

其余内容

可能还有一些特别的识别在以上没有展现，但是已经完成了书中要求的内容。