编译系统实验一

姓名：姚舜宇　学号：1190202107　班级：1903602

实验目的

通过使用词法分析工具 flex 和语法分析工具 bison，编写一个能够进行词法和语法分析的程序，能够检查出 C--代码中可能包含的词法错误和语法错误。

实验环境

GNU Linux Release: Ubuntu

GCC

GNU Flex

GNU Bison

实验内容

词法分析

正则表达式

Float：这里的正则表达式包含了科学计数法

```
/*float type*/
FLOAT ([+-]?([0-9]*\.?[0-9]+|[0-9]+\.))|([+-]?[0-9]*\.[0-9]*[eE][+-]?[0-9]+)
```

注释：包含了“//”和“/* */”两种类型。其中“/* */”是先匹配到“/*”后寻找最后一个“*/”

```
COMMENT ("//".*)|("/*"([*]*(([^*/])+([/])*)*)*"*/")
```

整型：

```
/*int type*/
INT_OCT [+-]?0[1-7][0-7]*
INT_DEC [+-]?0|[1-9][0-9]*
INT_HEX [+-]?0[xX][a-fA-F0-9]+
INT_BIN [+-]?0[bB][01]+
/*int errors*/
INT_HEX_ERROR [+-]?0[xX][a-fA-F0-9]*[g-zG-Z]+[a-fA-F0-9]*
INT_OCT_ERROR [+-]?0[0-7]*[89]+[0-7]*
INT_BIN_ERROR [+-]?0[bB][01]*[2-9]+[01]*
INT {INT_HEX}|{INT_DEC}|{INT_OCT}|{INT_BIN}|{INT_HEX_ERROR}|{INT_OCT_ERROR}|{INT_BIN_ERROR}
```

包含了对十进制、八进制、十六进制、二进制的匹配，以及对相应类型错误的匹配。

对应错误的匹配情况如下，会输出错误位置和原因。

```
{INT_HEX_ERROR} {printf("Error type A at line %d: Illegal hexadecimal number %s\n",yylineno,
yytext);}
{INT_OCT_ERROR} {printf("Error type A at line %d: Illegal octal number %s\n",yylineno,
yytext);}
{INT_BIN_ERROR} {printf("Error type A at line %d: Illegal binary number %s\n",yylineno,
yytext);}
```

语法分析

多叉树的构建

由于语法树为多叉树，故使用“儿子-兄弟”表示法。

节点定义：

```
typedef struct Node{
    int line;
    char* name;
    struct Node *fchild,*next;
    union{
        // id or int/float value
        char* id_type;
        int intval;
        float fltval;
    };
}* ST,* tnode;
```

其中定义了儿子节点和兄弟节点，以及联合体中的类型和值。
输出语法树时使用前序遍历 Preorder 函数。

```
void Preorder(ST st, int level)
{
    if (st != NULL)
    {
        if (st->line != -1)
        {
            for (i = 0; i < level; i++) printf("  ");
            printf("%s", st->name);
            if ((!strcmp(st->name, "ID")) || (!strcmp(st->name, "TYPE"))) printf(": %s", st-
>id_type);
            else if (!strcmp(st->name, "INT")) printf(": %d", st->intval);
            else if (!strcmp(st->name, "FLOAT")) printf(": %f", st->fltval);
            else printf("(%d)", st->line);
            printf("\n");
        }
        Preorder(st->fchild, level + 1);
        Preorder(st->next, level);
    }
}
```

错误恢复
为了使得程序查到一个词法或语法错误时，程序不会停止，故加入错误恢复。在定义产生式时加入错误情况。Error 后加入错误恢复符号，表示从恢复符号后继续运行分析程序。

```
VarDec:ID {$$=newST("VarDec",1,$1);nodeList[nodeNum]=$$;nodeNum++;}
        |VarDec LB INT RB {$$=newST("VarDec",4,$1,$2,$3,$4);nodeList[nodeNum]=$$;nodeNum++;}
        |error "\n" {yyerrok;yyclearin;}
        ;
```

```
Stmt:Exp SEMI {$$=newST("Stmt",2,$1,$2);nodeList[nodeNum]=$$;nodeNum++;}
        |Compst {$$=newST("Stmt",1,$1);nodeList[nodeNum]=$$;nodeNum++;}
        |RETURN Exp SEMI {$$=newST("Stmt",3,$1,$2,$3);nodeList[nodeNum]=$$;nodeNum++;}
        |IF LP Exp RP Stmt %prec LOWER_THAN_ELSE {$$=newST("Stmt",
5,$1,$2,$3,$4,$5);nodeList[nodeNum]=$$;nodeNum++;}
        |IF LP Exp RP Stmt ELSE Stmt {$$=newST("Stmt",
7,$1,$2,$3,$4,$5,$6,$7);nodeList[nodeNum]=$$;nodeNum++;}
        |WHILE LP Exp RP Stmt {$$=newST("Stmt",5,$1,$2,$3,$4,$5);nodeList[nodeNum]=$$;nodeNum+-
+;}
        |error SEMI {yyerrok;yyclearin;}
        ;
```

编译过程：
bison -d syntax.y
flex lexical.l
gcc syntax.tab.c lex.yy.c syntax_tree.c -lfl -o parser
./parser test.cmm

实验心得

通过使用词法分析工具和语法分析工具，完整编写了词法、语法分析代码，能够检查出C--代码中的错误，对词法分析的过程以及原理、语法中的产生式、错误恢复等有了进一步的了解。使用多叉树保存语法树，对多叉树的实现更加熟悉了。