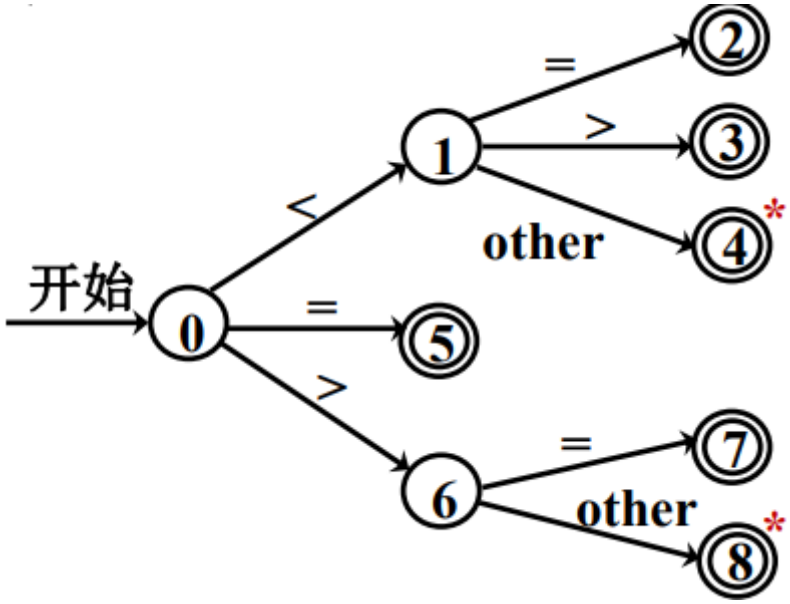
第一关

根据状态转换图：



编写getRelop()函数，用来读取关系运算符，并返回运算符种类。

然后在根据getRelop()函数返回的运算符种类调用transRelop()函数，将对应的运算符输出。

```
const char *transRelop(int Relop)
{
    switch (Relop)
    {
        case EQ:
            return "=";

        case LE:
            return "<=";

        case NE:
            return "<>";        // !=

        case L:
            return "<";

        case GE:
            return ">=";

        case G:
            return ">";

        default:
            return NULL;
    }
}
```

以上是在读取时对关系运算符的处理。

还有对其他字符的处理，若没有读取到换行符或者关系运算符就一直读下去，最后返回读取的字符长度。通过函数getWord()实现。

```
int getWord(char *Input, int *i)
{
    int length = 0;
    while (Input[*i] != '\0' && !isRelop(Input[*i]))    //一直读取直到读到关系运算符 or 换行符
    {
        if (Input[*i] != '\n' && Input[*i] != '\r')
        {
            length++;
        }
        (*i)++;
    }
    return length;  //返回字长度
}
```

最后在main()函数中进行输入，然后只需要循环调用getRelop()函数和getWord()函数即可，直到读取到换行符(输入结束)。

第二关

第二关是使用flex对输入流进行处理。

调用yylex()函数之后flex会自动地使用在relop.lex文件中定义的规则去匹配tokens，我们要做的只是定义规则以及决定在每个token被匹配后所要执行的动作。

根据实验要求，需要匹配的字符有'='，'<'，'>'，'<='，'>='，'<>'，'\n'，'\r'，以及由其他字符组成的字符串。

对应的动作为：匹配到关系符s则输出(relop,s)，匹配到'\n'或'\r'则结束读取，匹配到其他字符串则输出(other,len)，其中len为字符串的长度。

根据lex的语法，以上功能的实现如下：

```
"<"     {printf("(relop,<)");}
"<="    {printf("(relop,<=)");}
"="     {printf("(relop,=)");}
">"     {printf("(relop,>)");}
">="    {printf("(relop,>=)");}
"<>"    {printf("(relop,<>)");}
[^<>=\r\n]+ {printf("(other,%d)", yyleng);}
[\r\n]  {return 0;}
```

其中yyleng为匹配到的字符串的长度。

接下来只要对labLexer-2.c文件使用条件编译，当LEXERGEN定义时使用flex生成的分析器，未定义时使用第一关自己写的分析器即可。

当LEXERGEN定义时只需调用yylex()函数即可，flex就会自动进行分析并执行相关动作。未定义时与第一关相同。

labLexer-2.c文件的main()函数的代码如下：

```
int main()
{
```

```
#ifdef LEXERGEN
    flex_analyze(); //yylex()封装在该函数中
    return 0;
#else
    int i = 0;  //loop variable
    char *Input = (char *)malloc(sizeof(char) * MAX_LENGTH);
    memset(Input, '\0', MAX_LENGTH * sizeof(char));     // initialize
    fgets(Input, MAX_LENGTH, stdin);    // get input

    while (Input[i] != '\0')    //循环读取，直到到达数组末尾
    {
        int length = getWord(Input, &i);
        if (length != 0)
        {
            printf("(other,%d)", length);
        }
        int relop = getRelop(Input, &i);
        if (relop != 0)
        {
            printf("(relop,%s)", transRelop(relop));
        }
    }

    free(Input);

    return 0;
#endif
}
```

第三关

第三关是使用antlr对输入流进行自动处理。

基本操作、思路与flex的使用相似，需要在relop.g4文件中对定义匹配规则，以及决定对匹配的tokens所要做的操作。

根据实验要求，需要匹配的字符有'='，'<'，'>'，'<='，'>='，'<>'，'\n'，'\r'，以及由其他字符组成的字符串。

对应的动作为：匹配到关系符s则输出(relop,s)，匹配到'\n'或'\r'则结束读取，匹配到其他字符串则输出(other,len)，其中len为字符串的长度。

根据antlr的语法规则，relop.g4编写如下：

```
lexer grammar relop;    //词法分析器

@lexer::members {/* public lexer declarations section */
void PrintRelop(const char *relop)
{
    printf("(relop,%s)", relop);
    return;
};
}

tokens {
    Equal,
    NonEqual,
```

```
    Less,
    Greater,
    LessEqual,
    GreaterEqual,
    ID,
    WS
}
fragment Other: ~[<>=\r\n] ;

Equal: '='          {PrintRelop("=");};
NonEqual: '<>'      {PrintRelop("<>");};
Less: '<'           {PrintRelop("<");};
Greater: '>'        {PrintRelop(">");};
LessEqual: '<='     {PrintRelop("<=");};
GreaterEqual: '>='  {PrintRelop(">=");};
ID: Other+          {std::cout << "(other,"; std::cout <<
getText().length(); std::cout << ")";} ;
WS: [\r\n] -> skip ;

```

其中getText()函数返回匹配的字符串，返回类型是string对象，因此要取得字符串长度需要调用length()方法。

labLexer-3.cpp编写如下：

```
#include <iostream>

#include "antlr4-runtime.h"
#include "relop.h"
#define MAX_LENGTH 100

using namespace antlr4;

int main(int argc, const char* argv[])
{
    char Input[MAX_LENGTH];
    fgets(Input, MAX_LENGTH, stdin);
    ANTLRInputStream input(Input);
    relop lexer(&input);
    CommonTokenStream tokens(&lexer);

    tokens.fill();
//    for (auto token : tokens.getTokens()) {
//        std::cout << token->toString() << std::endl;
//    }

    return 0;
}

```

其中使用fgets()函数实现用户输入，ANTLRInputStream用于将输入的字符送入ANTLR词法分析器进行分析。接下来使用relop、CommonTokenStream将输入的字符流分割成一个个token并记录下来。最后调用tokens.fill()执行在relop.g4中定义的每个token后面的动作。