词法分析文档

设计目的

- 实现对 minic 文件转换为一个个 token 并生成词法树

实现方法

- lex

代码设计

c 语言部分定义

```
    %{
    #include "main.h"
    #include "parse.tab.h"
    /**
    * @description:  判断是否为关键字，如果不是则返回类型为ID
    * @param {void}
    * @return: int
    * @author: David.Huangjunlang
    */
    int isReservedWords();

    /**
    * @description: 打印词法树
    * @param {int}
    * @return: void
    * @author: David.Huangjunlang
    */
    void lexAnalysis(int index);
    // 链接词法输出文件
    extern FILE* lexOut;
    extern "C"{
        int yywrap(void);
    }
    char c[4][10]={"ID","NUM","RESERVED","OPERATOR"};
%}
```

- main.h 头文件中包含了所需要的数据结构的定义

```
    typedef struct
    {
        std::string m_id; // string 类型存储id 的名字
        std::string m_reserved; // string 类型存储关键字
        int m_num; // int 类型 存储常量值
        std::string m_op; // string 类型 存储符号
        node *m_node; // node 存储语句
    }   ytype;
```

- parse.tab.h 头文件有 yacc parse.y 生成，包含了定义好的返回 token 的值

```
/* Tokens.  */
#ifndef YYTOKENTYPE
```

```
# define YYTOKENTYPE
   /* Put the tokens into the symbol table, so that GDB and other debuggers
      know about them.  */
   enum yytokentype {
     ID = 258,
     NUM = 259,
     PLUS = 260,
     MINUS = 261,
     MULTI = 262,
     DIVIDE = 263,
     LESS = 264,
     LESSEQUAL = 265,
     GREATER = 266,
     GREATEREQUAL = 267,
     EQUAL = 268,
     UNEQUAL = 269,
     ASSIGNMENT = 270,
     SEMICOLON = 271,
     COMMA = 272,
     LEFTBRACKET = 273,
     RIGHTBRACKET = 274,
     LEFTSQUAREBRACKET = 275,
     RIGHTSQUAREBRACKET = 276,
     LEFTBRACE = 277,
     RIGHTBRACE = 278,
     ELSE = 279,
     IF = 280,
     INT = 281,
     VOID = 282,
     RETURN = 283,
     WHILE = 284
   };
#endif
```

- isReservedWords() 函数用于判断该 token 是 id 还是保留字

```
/**
 * @description:  判断是否为关键字，如果不是则返回类型为ID
 * @param {void}
 * @return: int
 * @author: David.Huangjunlang
 */
int isReservedWords(){
        if(strcmp(yytext,"else") == 0){yylval.m_reserved =
yytext;lexAnalysis(2);return ELSE;}
        if(strcmp(yytext,"if") == 0){yylval.m_reserved =
yytext;lexAnalysis(2);return IF;}
        if(strcmp(yytext,"int") == 0){yylval.m_reserved =
yytext;lexAnalysis(2);return INT;}
        if(strcmp(yytext,"return") == 0){yylval.m_reserved =
yytext;lexAnalysis(2);return RETURN;}
        if(strcmp(yytext,"void") == 0){yylval.m_reserved =
yytext;lexAnalysis(2);return VOID;}
        if(strcmp(yytext,"while") == 0){yylval.m_reserved =
yytext;lexAnalysis(2);return WHILE;}
        yylval.m_id = yytext;
        lexAnalysis(0);
```

```
        return ID;
}
```

- lexAnalysis(int index) 函数用于输出词法树,参数为所属类型的 index,各类型保存在 char c[][]中, lexOut 为输出的目标文件，在程序运行时已定义所以使用 extern

```
    char c[4][10]={"ID","NUM","RESERVED","OPERATOR"};
/**
 * @description: 打印词法树
 * @param {int}
 * @return: void
 * @author: David.Huangjunlang
 */
void lexAnalysis(int index){
    fprintf(lexOut, "< %-6d, %-10s, %-7s>\n", yylineno, c[index], yytext);
}
```

- yywarp 为 lex 所需要的 c 语言的函数

lex 全局定义

- COMMENT 用于识别注释内容，其余为 数字 字母 空格 换行符 id 的定义

```
%x          COMMENT
digit       [0-9]
letter      [a-zA-Z]
delim       [" "\t]
nexLine     [\n]
whitespace  {delim}+
ID          {letter}+
NUM         {digit}+
%%
```

lex 模式匹配规则

- id 识别，识别后利用 isReservedWords() 函数判断是否为保留字

```
{ID}            {return isReservedWords();}
```

- NUM(数字) 识别，识别后转换为 int 类型，并返回

```
{NUM}           {yylval.m_num = atoi(yytext);lexAnalysis(1);return NUM;}
```

- 各种符号的识别并保存符号的字符串后返回

```
{whitespace}    {}
{nexLine}       {yylineno++;}
"+"             {yylval.m_op = yytext;lexAnalysis(3);return PLUS;}
"-"             {yylval.m_op = yytext;lexAnalysis(3);return MINUS;}
"*"             {yylval.m_op = yytext;lexAnalysis(3);return MULTI;}
"/"             {yylval.m_op = yytext;lexAnalysis(3);return DIVIDE;}
"<"             {yylval.m_op = yytext;lexAnalysis(3);return LESS;}
"<="            {yylval.m_op = yytext;lexAnalysis(3);return LESSEQUAL;}
">"             {yylval.m_op = yytext;lexAnalysis(3);return GREATER;}
">="            {yylval.m_op = yytext;lexAnalysis(3);return GREATEREQUAL;}
```

```
"=="             {yylval.m_op = yytext;lexAnalysis(3);return EQUAL;}
"!="             {yylval.m_op = yytext;lexAnalysis(3);return UNEQUAL;}
"="              {yylval.m_op = yytext;lexAnalysis(3);return ASSIGNMENT;}
";"              {yylval.m_op = yytext;lexAnalysis(3);return SEMICOLON;}
","              {yylval.m_op = yytext;lexAnalysis(3);return COMMA;}
"("              {yylval.m_op = yytext;lexAnalysis(3);return LEFTBRACKET;}
")"              {yylval.m_op = yytext;lexAnalysis(3);return RIGHTBRACKET;}
"["              {yylval.m_op = yytext;lexAnalysis(3);return LEFTSQUAREBRACKET;}
"]"              {yylval.m_op = yytext;lexAnalysis(3);return RIGHTSQUAREBRACKET;}
"{"              {yylval.m_op = yytext;lexAnalysis(3);return LEFTBRACE;}
"}"              {yylval.m_op = yytext;lexAnalysis(3);return RIGHTBRACE;}
```

- 注释的识别, 识别到注释后进入 COMMENT 状态，并在识别到 注释结束符后再重新进入开始状态

```
"/*"             {BEGIN COMMENT;}
<COMMENT>"*/"    {BEGIN INITIAL;}
```

测试数据

```
void hello(void){return;}
/* hello*/


int gcd(int u, int v){
    if (v == 0)return u;
    else return gcd(v, u-u/v*v);
    /* test */
}

void main(void){
    int x, int y;
    x = input();
    y = input();
    output(gcd(x, y));
}
```

测试结果

```
< 1     , RESERVED  , void   >
< 1     , ID        , hello  >
< 1     , OPERATOR  , (      >
< 1     , RESERVED  , void   >
< 1     , OPERATOR  , )      >
< 1     , OPERATOR  , {      >
< 1     , RESERVED  , return >
< 1     , OPERATOR  , ;      >
< 1     , OPERATOR  , }      >
< 5     , RESERVED  , int    >
< 5     , ID        , gcd    >
< 5     , OPERATOR  , (      >
< 5     , RESERVED  , int    >
< 5     , ID        , u      >
< 5     , OPERATOR  , ,      >
< 5     , RESERVED  , int    >
< 5     , ID        , v      >
```

```
< 5     , OPERATOR  , )      >
< 5     , OPERATOR  , {      >
< 6     , RESERVED  , if     >
< 6     , OPERATOR  , (      >
< 6     , ID        , v      >
< 6     , OPERATOR  , ==     >
< 6     , NUM       , 0      >
< 6     , OPERATOR  , )      >
< 6     , RESERVED  , return >
< 6     , ID        , u      >
< 6     , OPERATOR  , ;      >
< 7     , RESERVED  , else   >
< 7     , RESERVED  , return >
< 7     , ID        , gcd    >
< 7     , OPERATOR  , (      >
< 7     , ID        , v      >
< 7     , OPERATOR  , ,      >
< 7     , ID        , u      >
< 7     , OPERATOR  , -      >
< 7     , ID        , u      >
< 7     , OPERATOR  , /      >
< 7     , ID        , v      >
< 7     , OPERATOR  , *      >
< 7     , ID        , v      >
< 7     , OPERATOR  , )      >
< 7     , OPERATOR  , ;      >
< 9     , OPERATOR  , }      >
< 11    , RESERVED  , void   >
< 11    , ID        , main   >
< 11    , OPERATOR  , (      >
< 11    , RESERVED  , void   >
< 11    , OPERATOR  , )      >
< 11    , OPERATOR  , {      >
< 12    , RESERVED  , int    >
< 12    , ID        , x      >
< 12    , OPERATOR  , ,      >
< 12    , RESERVED  , int    >
< 12    , ID        , y      >
< 12    , OPERATOR  , ;      >
< 13    , ID        , x      >
< 13    , OPERATOR  , =      >
< 13    , ID        , input  >
< 13    , OPERATOR  , (      >
< 13    , OPERATOR  , )      >
< 13    , OPERATOR  , ;      >
< 14    , ID        , y      >
< 14    , OPERATOR  , =      >
< 14    , ID        , input  >
< 14    , OPERATOR  , (      >
< 14    , OPERATOR  , )      >
< 14    , OPERATOR  , ;      >
< 15    , ID        , output >
< 15    , OPERATOR  , (      >
< 15    , ID        , gcd    >
< 15    , OPERATOR  , (      >
< 15    , ID        , x      >
< 15    , OPERATOR  , ,      >
< 15    , ID        , y      >
```

```
< 15    , OPERATOR  , )      >
< 15    , OPERATOR  , )      >
< 15    , OPERATOR  , ;      >
< 16    , OPERATOR  , }      >
```