编译原理实验报告

1110310123
徐伟

目录

实验题目

编译原理实验

实验目的

1.词法分析器的设计与实现 ；
2.语法分析器的设计与实现 ；
3.语义分析和中间代码生成器的设计与实现 。

实验内容和要求

1. 词法分析部分

   a) 内容及概述

   语言包含关键字、标识符、运算符、分隔符、常数以及注释。
   关键字包括：

```
int , double , char , bool , return , void , if , else , for , break , continue , true , false ,
printf , scanf , main
```

   标识符的定义为：
   (letter |'_')(letter | digit |'_')* ;
   运算符包括：
   > , < , = , <= ,>= , == , != , ! , (,) , ++ , -- , && , || ;
   分隔符包括：
   ',' , ';' , '{' , '}' ,'(' , ')' ;
   常数包括：
   整型(INT_C) ， 定义为 (digit)* ;
   实数型(REAL_C) ， 定义为 (digit)*.(digit)* ;
   字符型(CHAR_C) ， 定义为 'letter' ;
   字符串型(STRING_C ， 定义为 双引号包含的字符串；
   布尔常量(BOOL_C) ， 定义为 true , false ;
   注释包括：
   "//" 单行注释
   "/* */ "多行注释

   b) 词法分析阶段出错处理
      1. 非法字符报错但不结束分析
      2. 小括号、大括号不匹配报错但不结束分析

2. 语法分析部分

   a) 内容及概述

语法分析首先定义了终结符、非终结符、产生式、文法开始符号。之后根据上述内容构建了 LR1 分析表。分析表构建完成后根据 LR 分析法移进规约原则进行语法分析。

终结符定义为：

```
INT   CHAR DOUBLE    BOOL RETURN    VOID IF   ELSE      FOR       BREAK
CONTINUE   TRUE FALSEMAIN PRINTF    SCANFID   REAL_C    INT_CCHAR_C         STRING_C
GE    GT   LE   LT   EQ   ASSIGN    NEQ       NOT       LR_BRAC
RR_BRAC         LS_BRAC         RS_BRAC         INC  ADD       DEC       SUB
MUL_OR_INDIR    DIV       AND       REFERENCE OR        COMMA     SEMIC
LB_BRAC         RB_BRAC
```

非终结符定义为：

```
P              X              mainFun        funDefine      funHead    funBody
funDomain      declaration          assignment     condition      loop       io
funApply       returnWord     jumpWord       paramList      type       idList
singleOP       EXP            E              T              F          DIGIT
G              basicBooleanEXP               relOP          logicOP    acturalParam
forAssignPart  forBoolPart    forAssignList  forAssignment  returnVal  printContent
returnType     formalParam
```

产生式定义为

```
P => X ;
X -> funDefine X | mainFun ;
funDefine -> funHead funBody ;
funHead -> returnType ID LR_BRAC formalParam RR_BRAC ;
funBody -> LB_BRAC funDomain RB_BRAC ;
funDomain ->   declaration funDomain |
               assignment funDomain |
               condition funDomain |
               loop funDomain |
               io funDomain |
               funApply funDomain |
               jumpWord funDomain |
               SEMIC funDomain |
                ;
returnType -> INT | DOUBLE | CHAR | BOOL | VOID ;
formalParam -> paramList | ;
paramList -> type ID COMMA paramList | type ID ;
type -> INT | DOUBLE | BOOL | CHAR ;
declaration -> type idList SEMIC ;
idList -> ID COMMA idList | ID ;
assignment ->  ID ASSIGN EXP SEMIC |
          ID ASSIGN funApply |
```

```
            singleOP SEMIC ;
    EXP -> E AND EXP | E OR EXP | E ;
    E -> T ADD E | T SUB E | T ;
    T -> F MUL_OR_INDIR T | F DIV T | F ;
    F -> G relOP G | G ;
    G -> LR_BRAC EXP RR_BRAC | ID | TRUE | FALSE | DIGIT | CHAR_C ;
    DIGIT -> INT_C | REAL_C ;
    relOP -> EQ | NEQ | GE | GT | LE | LT ;
    funApply -> ID LR_BRAC acturalParam RR_BRAC SEMIC ;
    acturalParam -> idList | ;
    singleOP -> INC ID | ID INC | DEC ID | ID DEC ;
    condition -> IF LR_BRAC EXP RR_BRAC funBody |
                 IF LR_BRAC EXP RR_BRAC funBody ELSE funBody ;
    loop -> FOR LR_BRAC forAssignPart SEMIC forBoolPart SEMIC forAssignPart RR_BRAC funBody ;
    forAssignPart -> forAssignList | ;
    forAssignList -> forAssignment COMMA forAssignList | forAssignment ;
    forAssignment -> ID ASSIGN EXP | singleOP ;
    forBoolPart -> EXP | ;
    jumpWord -> returnWord | CONTINUE SEMIC | BREAK SEMIC ;
    returnWord -> RETURN returnVal SEMIC ;
    returnVal -> EXP | ;
    io -> PRINTF LR_BRAC printContent RR_BRAC SEMIC | SCANF LR_BRAC STRING_C COMMA REFERENCE
ID RR_BRAC SEMIC ;
    printContent -> STRING_C | STRING_C COMMA ID ;
    mainFun -> INT MAIN LR_BRAC formalParam RR_BRAC funBody ;
```

文法开始符：

```
    P
```

终结符即是包含词法分析中的关键字、运算符、分隔符、常数等。

非终结符是文法中各种语法成分。

产生式是根据 C 的文法、需要实现的功能、规模控制等综合考虑做出来的。其中包含了要求实现的语句：

说明语句　(对应产生式的 declaration)；

赋值语句　(对应产生式的 assignment)；

逻辑表达式和算术表达式　(对应产生式的 EXP)；

顺序语句　(对应产生式的 funDomain)；

IF 语句　(对应产生式的 condition)；

循环语句　(对应产生式的 loop)；

过程说明语句　(对应产生式的 funDefine)；

过程调用语句　(对应产生式的 funApply)；

输入输出语句　(对应产生式的 io)。

在完成上述的定义后，接着完成 LR1 分析表的构建 。 首先读取上述定义，在

内存中将各终结符、非终结符、结束符#映射为从 0 开始的连续数字作为分析表的横坐标。然后从第一个 LR1 项目[P->X , #]开始，求得该项目的 LR1 项目集闭包，即是 I_0 项目集族。然后遍历 I_0 中所有项目，求得每个项目的后继项目，合并读入字符相同的后继项目，然后对每个项目求其项目集闭包，若该项目集闭包和已有的项目集闭包不重复，则加入到项目集族集合中。以此循环，直到项目集族不在增大，则整个 LR1 项目集族建立完成。在构建项目集族中，还需要记录 GO(I , X)的信息用来填写 ACTION 表中的移进动作和 GOTO 表中的转移。此外，对每个项目集族中出现的规约项目，还需要记录到规约记录集中，用来填写 ACTION 表中的规约动作项。

在完成上述操作后，我们就可以构建 LR1 分析表了。在 ACTION 表中，另大于等于 0 的数表示移进转移动作，该数值即为转移到的状态号，即分析表的纵坐标；小于 0 的数表示规约动作，该数值的相反数即表示规约所用的产生式编号(由于不会用第 0 号产生式，即拓展文法的开始项，来规约，故不会有冲突)。用+Infinity 表示 ACC ，用-Infinity 表示 ERROR 。读取上述建立的两个数据记录，就完成了分析表的构建。

分析表构建完成后，建立符号栈和状态栈，按照教材上的算法移进规约即可，不再赘述。

b) 语法分析的出错处理

读取到 ERROR 时强制退出，语法分析失败，待分析代码不符合文法定义

3. 语义分析部分

a) 内容及概述

首先分析产生式，根据自底向上的语义分析方法，在 funHead , condition , loop , mainFun 中加入 ACT_1,ACT_2 等 6 个标记非终结符。具体如下：

```
    funHead -> returnType ID ACT_1 LR_BRAC formalParam RR_BRAC ;
    condition -> IF LR_BRAC EXP ACT_2 RR_BRAC funBody |
                 IF LR_BRAC EXP ACT_2 RR_BRAC funBody ACT_3 ELSE funBody ;
    loop -> FOR LR_BRAC forAssignPart ACT_4 SEMIC forBoolPart ACT_5 SEMIC forAssignPart ACT_6
RR_BRAC funBody ;
    mainFun -> INT MAIN ACT_1 LR_BRAC formalParam RR_BRAC funBody ;
    ACT_1 -> ;
    ACT_2 -> ;
    ACT_3 -> ;
    ACT_4 -> ;
    ACT_5 -> ;
    ACT_6 -> ;
```

这些动作主要是用于创建新的符号表，操纵符号表栈、offset 栈、标签栈，输出标签等。

为了实现语义分析，首先加入语义栈，该语义栈记录词法分析的信息，在语法分析移进操作时同时将该符号的信息压入语义栈。然后是建立符号表，offset，为了管理嵌套，建立符号表栈，offset 栈。后来为了在 if、for、break、continue、return

等跳转语句中输出相应的位置，再建立了标签和标签栈。

由以上分析，语义分析就是在合适的时间正确的操纵上述各个栈和数据。

具体操作方法见后文。

b) 语义分析出错处理

处理的错误有：

1. 类型自动转换失败。若类型转换将导致数据丢失(如将 int 转为 char)时出错。
2. 引用未声明变量。
3. 错误的 break、continue 语句(即在无循环的作用域内使用 break、continue)。
4. 将 void 类型的函数返回值赋给变量。

处理方法为：

打印错误信息，结束分析。

总体设计

1. 总体结构

分为词法、语法、语义、主函数四部分。

词法部分完成对源文件的词法分析。输出为 lexType ， lexVal ；的二元组形式。该输出作为语法分析的输入。这些二元组信息将同时会被压入到语义分析的语义栈中。

语法分析读取终结符、非终结符、产生式，建立 LR1 分析表，对词法分析的输出处理，同时在移进时操作符号栈、状态栈和语义栈。在规约时调用语义分析的动作。

语义分析是在语法分析时被调用的。在移进时需要将词法分析的二元组移进到语义栈；在规约时需要根据规约的产生式编号进行相应的语义动作。

主函数调用各模块。

2. 数据结构设计

a) 符号表

```
#define TB_NODE_MAX_SIZE 200
struct tbNode
{
    char name[20] ;
    char type[16] ;
    void * addr ;
} ;
struct symbolTb
{
    struct symbolTb * previous ;
    int width ;
    int counter ;
    struct tbNode * data[TB_NODE_MAX_SIZE] ;
} ;
```

symbolTb 结构体即是一个符号表定义，包含指向其父符号表的指针 previous ，符号表中所存变量的总大小 width ， 符号表中存储变量的个数 counter ， 保存

变量信息的数组 data 。其中 data 是一个结构体数组，类型为 tbNode ，该类型包含 name 、 type 、 addr 属性。

由于需要支持作用域嵌套，故在创建新的作用域时，需要保存父符号表的指针，通过 previous 指针，就能够在该作用域下通过该指针访问上级作用域了。对于顶级作用域值，该值为空。存储分配时，需要知道需要的空间大小，故需要 width 属性。由于是数组存储，需要知道存储的个数，故设立 counter 属性。tbNode 中，name 用于存储变量名，是唯一标识，需要在插入前进行检查。type 记录类型，addr 记录地址。这是常规变量的使用。由于新的子符号表也要放到该符号表项中，故当存储的不是常规变量而是符号表时，设置 name 属性为符号表名（如函数，则为函数名，若为 for、if 等作用域，则建立临时名字即可），type 设置为 TABLE + returnType。即 type 属性设置为 TABLE，同时在其后附加上返回值，这主要是用于对函数的返回值检查。当存在将函数返回值赋给一个变量时，就需要查表，获取 type 中函数返回值字段，确定该赋值是否合法。addr 存储新符号表的地址。这也是为何 addr 设置为 void * 的原因，因为它即可能是普通变量的地址（其实就是一个相对偏移），也可以是符号表的地址。

b) 关键字

```
#define KEYWORD_NUM 500
#define KEYWORD_NUM_PRIME 499
typedef struct Word
{
    char name[15] ;
    struct Word * nxt ;
}Word ;
typedef struct KeyWord
{
    Word * words[KEYWORD_NUM] ;
}KeyWord ;
```

KeyWord 结构体是关键字定义，包含的 words 属性是 Word 结构体指针数组，Word 包含关键字名称 name，和指向下一个关键字的 nxt 指针。

由于需要频繁的查询读入的标识符是否是关键字，所以需要要求查询快速。于是采用 hash 方法。

```
int insertKeyWord(KeyWord * keyWord , char name[])
{
    int index = getHashCode((char *)name) % KEYWORD_NUM_PRIME ;
    if(keyWord->words[index] == NULL)
    {
        keyWord->words[index] = (Word *)malloc(sizeof(Word)) ;
        keyWord->words[index]->nxt = NULL ;
        strcpy(keyWord->words[index]->name,name) ;
        /* capitalize*/
        return index ;
    }
    else
    {
```

```
            Word * pos = keyWord->words[index] ;
            while(pos->nxt != NULL)
            {
                pos = pos->nxt ;
            }
            pos->nxt = (Word *)malloc(sizeof(Word)) ;
            pos = pos->nxt ;
            strcpy(pos->name,name) ;
            pos->nxt = NULL ;
            return -index ;
        }
    }
```

通过教材中的 hashpjw 函数对传入的标识符求得散列值，然后插入到关键字表。若出现冲突，则插入到 nxt 指向的位置。关键字比较少，但是将关键字数组开这么大，就是为了减少冲突的产生，提高查询速度。

c) 双缓冲

```
#define DBUF_SIZE (1024*2)
typedef struct DBuffer
{
    char * buf ;
    int head ;
    int rear ;
    int hasBack ;
    FILE * fp ;
}DBuffer ;
```

该数据结构是用来实现双缓冲机制的。初始化缓冲时，buf 被分配给 DBUF_SIZE 大小的缓冲区，缓冲区被分成两部分，在中间和缓冲区结束均以 EOF 填充。以 FP 关联需要缓冲的文件。初始时读入 DBUF_SIZE/2 -1 个字符到双缓冲区前部分。然后造读操作时，若读到 EOF，则再读取文件到双缓冲的后半部分。若读到后半部分的 EOF，则读取文件到缓冲区的前半部分，以此类推。head , rear 用来标识缓冲区的单词起止位置，copyToken 函数就利用这两个位置完成单词复制。同时，由于词法分析存在回退，考虑一种情况，即是当缓冲区刚刚读到 EOF，从文件载入了内容到缓冲区，此时回退一个字符，又会读到 EOF，若不加判断，则又会从文件载入内容到缓冲区，这样半个缓冲区的内容就被丢失了！故加入 hasBack 标识位来处理这个问题。初始 hasBack 为假(0),当 hasBack 为假时，读到 EOF 载入文件到缓冲；若有回退操作，则 hasBack 为真(1)，此时若紧接着读到 EOF，则忽略，不再读取文件到缓冲，同时将 hasBack 置为假。通过这些设置，就完成了双缓冲区，同时封装了 readyCopy , copyToken , retract 操作。具体代码实现见源代码。

d) 符号映射表：数字 -> 符号

```
#define TERMINAL 0
#define NON_TERMINAL 1
typedef struct SymbolNumStandsFor
{
    char name[SYMBOL_NAME_MAX_LEN] ;
```

```
    short type ;
    /* attention ! the productionIndex[0] stands for the number of the producion*/
    int productionIndex[11] ;
} SymbolNumStandsFor ;
/** symbols list */
struct SymbolList
{
    SymbolNumStandsFor data[SYMBOL_MAX_NUM] ;
    int symbolNum ;
}
symbolList ;
```

symbolList 是将数组下标作为索引，求得该索引对应的符号项。该符号项即 SymbilNumStandsFor 结构体，包含符号值 name，符号类型（宏定义的 TERMINAL 和 NON_TERMINAL），以及以该符号为左部的产生式编号 productionIndex 数组。该数组第一个值用来表示以该符号为左部的产生式个数，而后挨个表示产生式编号。该编号的映射关系在后面介绍。

e) 符号映射表：符号 -> 数字

```
#define SYMBOL_HASH_LEN 457
#define SYMBOL_NAME_MAX_LEN 25
typedef struct SymbolNumExp
{
    int numExp ;
    char name[SYMBOL_NAME_MAX_LEN] ;
} SymbolNumExp ;
SymbolNumExp transTable[SYMBOL_HASH_LEN] ;
```

这是上一个符号映射表的基础。在文件中存储的是字符形式的符号，读入到内存后在后续处理中符号都要被转换为数字，且所有符号的数字表示合起来是从 0 到 symbolNum-1 的连续数字，即以此作为 analysisTable 分析表的横坐标。这里为了速度同样采用了 hash 的方式。

定义 i = 0 ；首先从 terminallist.txt 中读取终结符，转换为 hash 索引，在该索引处的 SymbolNumExp 中的 name 设为该符号值，将 numExp 设为 i，同时填写上面给的从数字到字符的字符映射表，即第 i 项的 name 值设为符号值，type 设为 TERMINAL，productionIndex[0] = 0 ;然后 i++ ，处理下一个；完成后再将结束符#加入进去。最后再读 nonterminallist.txt 文件，相同的方式处理。最后读完后，数字和字符的双向映射就做好了。在这之上封装了 tansName() ，getName()函数来读取这种映射关系。具体代码实现见源码中 analysis_table.c 。

f) 产生式

```
#define PRODUCTION_MAX_LEN 15
typedef struct Production
{
    int pLeft ;
    int pRight[PRODUCTION_MAX_LEN] ;
    int order ;/* the order of the production */
    int len ; /* the length of the production */
```

```
} Production ;


/** production */
typedef struct SProduction
{
    Production data[PRODUCTION_MAX_NUM] ;
    int productionNum ;
} SProduction ;
SProduction productions ;
```

由之前的基础，读取 production.txt，数字化所有的产生式到该 produtions 结构体即可。pLeft 为数字表示的产生式左部，pRight 数组为右部，len 标识了右部长度。order 反向表示了该产生式的编号，即是该产生式在该结构体数组的下标。

在这之上封装了 restoreP 函数，用来将产生式标号转换为字符型产生式。

g) LR1 项目

```
#define EXPECTED_SYMBOL_MAX_LEN 30
typedef struct LR1_Item
{
    int productionOrder ;
    int dotPos ;
    int expectedSymbol[EXPECTED_SYMBOL_MAX_LEN] ;
    int exSymNum ;/* the expected symbol num */
    struct LR1_Item * next ;
} LR1_Item ;
```

包含 LR1 项目需要的全部内容，productionOrder 表示产生式编号，dotPos 表示小圆点位置，即是状态。expectedSymbol 和 exSymNum 用来表示数字化的展望符集合和展望符个数。next 是用来表示项目集族的。下一个会用到。

h) LR1 项目集族表示

```
typedef struct ItemCollection
{
    int len ;
    int order ;
    short hasReducedItem ;
    LR1_Item * items ;
} ItemCollection ;
```

len 表示该项目集族的包含 LR1 项目个数，order 表示项目集族编号，hasReducedItem 用来表示该集族是否含有规约项目。items 用链表的形式存储该集族中所有项目。len 可用于快速判读两个项目集族是否相等，order 在建 Golist 表时需要。

该项目集族组合成一个项目集族结构体：

```
/** itemCollections */
#define STATE_MAX_NUM 1000
typedef struct SItemCollection
{
    ItemCollection * data[STATE_MAX_NUM] ; /* conveniente*/
    int icNum ;
```

```
} SItemCollection ;
SItemCollection ic ;
```

即是状态集。

i) GOList 表

```
struct GOList
{
        int startState ;
        int gotoState ;
        int symbol ;
        struct GOList * next ;
} * goHead , * goRear ;
```

如前所述，用来填 analysis_table 的 GO(I,X)链表。startState 为当前状态，symbol 为读入的符号，包括终结符和非终结符(当然，此时应该叫做被规约的符号),gotoState 为转移到的状态号。通过链表来构建这个表，故有 next 项。

j) ReduceList 表

```
struct ReduceList
{
    int state ;
    LR1_Item * reducedItem ;
    struct ReduceList * next ;
} * reduceHead , * reduceRear ;
```

类似道理，不再赘述。

k) 分析表

```
#define SYMBOL_MAX_NUM 100
#define STATE_MAX_NUM 1000
typedef struct AnalysisTable
{
    int table[STATE_MAX_NUM][SYMBOL_MAX_NUM] ;
    int row ;
    int col ;
} AnalysisTable ;
AnalysisTable analysisTable ;
```

语法分析中，最核心的表，一切都是为了这个表。

l) 语义栈

```
#define SEMANTIC_STACK_DEPTH 1000
struct SemanticNode
{
    /* it is the type at lexical result */
    char lexType[25] ;
    /* it is the value at the lexical result */
    char lexVal[25] ;
    /* extend , for addr , for param num , the STRING_C addr */
    void * extend ;
} ;
```

```
struct SemanticStack
{
    struct SemanticNode data[SEMANTIC_STACK_DEPTH] ;
    int top ;
} semanticS ;
```

语义栈，语义分析的核心。注释很清晰了，不再赘述。

m) offset 栈

```
struct OffsetStack
{
    void * data[TB_MAX_NUM] ;
    int top ;
} offsetS ;
```

存储 offset 信息。

n) tbptr 栈

```
#define TB_MAX_NUM 40
struct TbptrStack
{
    struct symbolTb  * data[TB_MAX_NUM] ;
    int top ;
} tbptrS ;
```

符号表栈

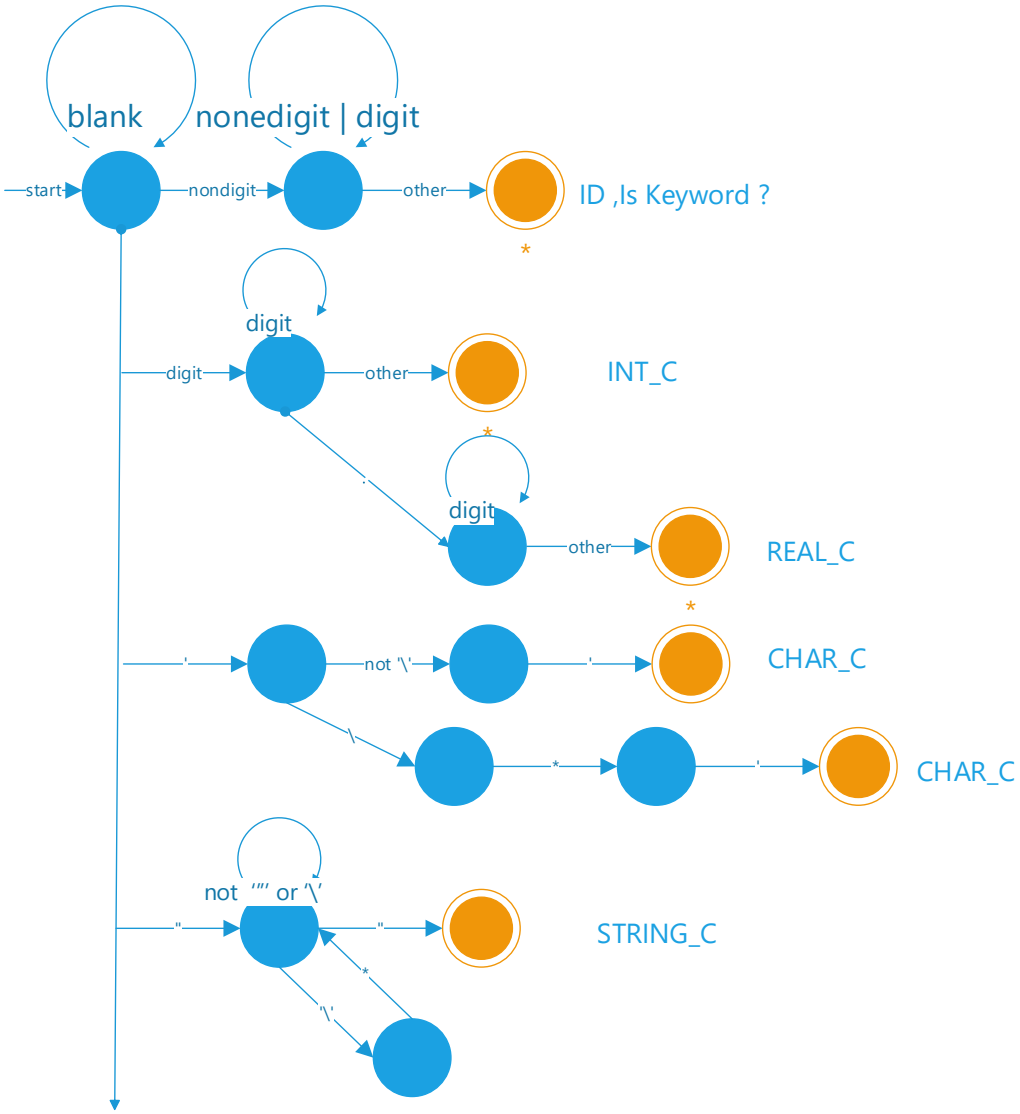
o) 标签、标签栈

```
/* Label */
struct LbS
{
    char data[LABEL_MAX_NUM][10] ; //that means max 9999 labels
    int top ;
} ;
struct LbptrStack
{
    struct LbS * data[TB_MAX_NUM] ;
    int top ;
} lbptrS ;
```

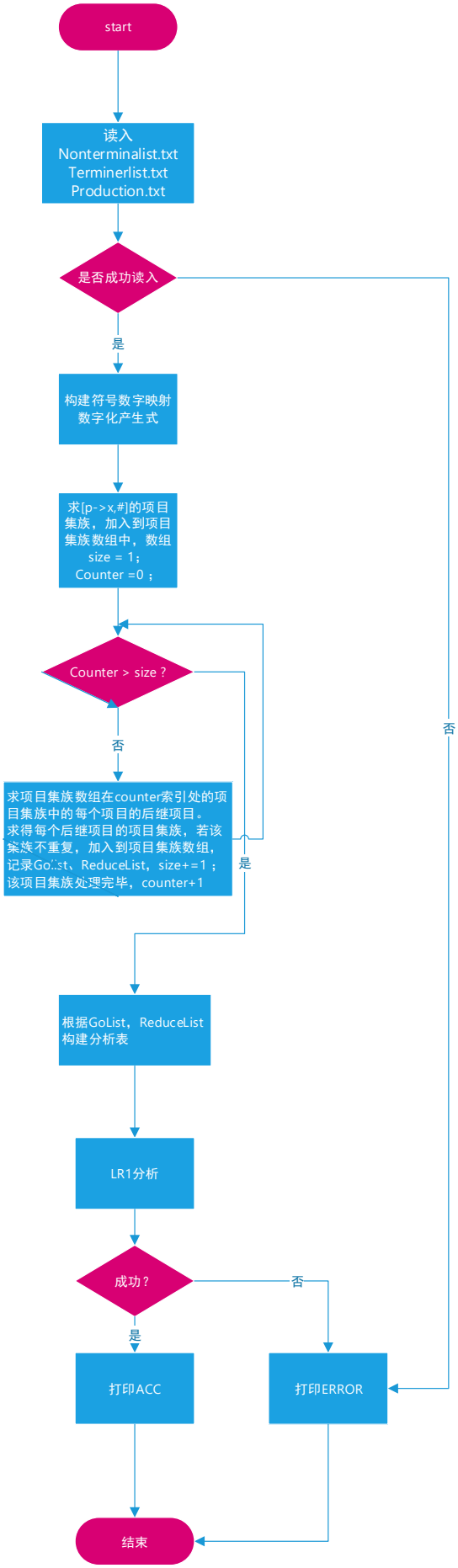
用来完成标签的输出。在每个函数建立前都会建立一个标签组，建立一个标签，同时压栈，在程序的末尾输出该标签，用于 return 的跳转。在单独 IF 中需要建立一个标签组，2 个标签；IF ELSE 结构需要建立一个标签组，3 个标签；FOR 语句需要建立一个标签组，4 个标签。在函数结束 ，IF、FOR 结束后需要将标签组弹栈、释放空间。

详细设计与实现

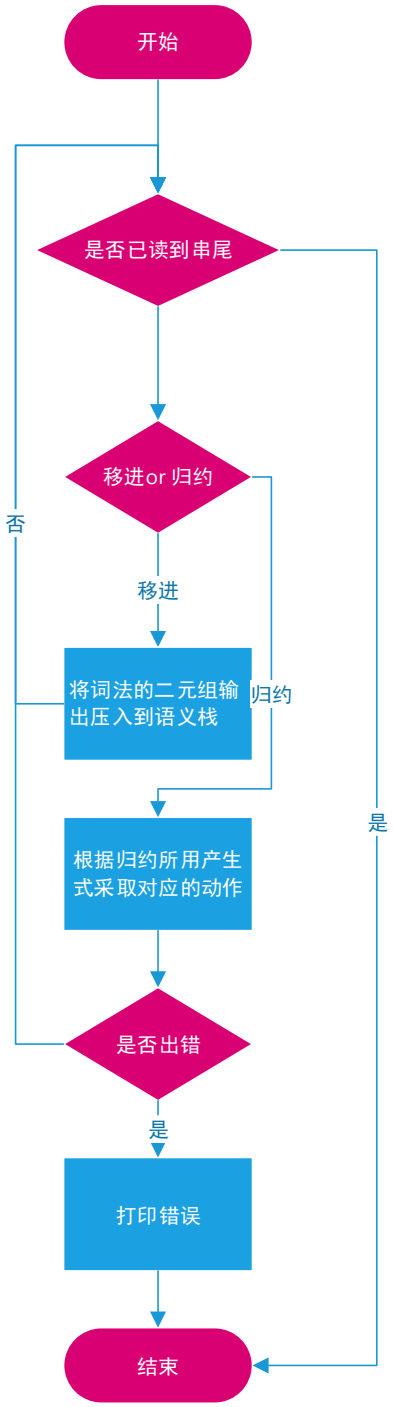
1. 词法分析的状态转换图：


blank
nonedigit | digit
start
nondigit
other
ID ,Is Keyword ?
*
digit
digit
other
INT_C
*
.
digit
other
REAL_C
*
'
not '\'
'
CHAR_C
\
*
'
CHAR_C
not '"' or '\'
"
"
STRING_C
*
"\'

2. 语法分析器程序框图


start
读入
Nonterminalist.txt
Terminerlist.txt
Production.txt
是否成功读入
是
否
构建符号数字映射
数字化产生式
求[p->x,#]的项目
集族，加入到项目
集族数组中，数组
size = 1；
Counter =0 ；
Counter > size ?
否
是
求项目集族数组在counter索引处的项
目集族中的每个项目的后继项目。
求得每个后继项目的项目集族，若该
集族不重复，加入到项目集族数组，
记录GoList、ReduceList，size+=1 ；
该项目集族处理完毕，counter+1
根据GoList，ReduceList
构建分析表
LR1分析
成功？
否
是
打印ACC
打印ERROR
结束

3. 语义分析程序框图



4. 具体实现(关键代码实现)

a) 词法

```
int lexAnalysis(char * srcName , char * outName)
{
    int errc = 0 ;
    char sourceFileName[20] ;
    strcpy(sourceFileName  , srcName) ;
    /** pre-process */
    char processedFileName[20] ;
```

```
        preProcess(sourceFileName,processedFileName) ;
       // printf("%s\n",fileName) ;
        /** init keyword */
        KeyWord * keyWord = (KeyWord *)malloc(sizeof(KeyWord)) ;
        FILE * keywordFile = fopen("keywords.txt","r") ;
        initKeyWord(keyWord) ;
        if(!keywordFile)
        {
            printf("Init keyword error! \n") ;
            return -1 ;
        }
        insertKeyWordFromFile(keyWord,keywordFile) ;
        /** init the stack */
        Stack * stack = (Stack *)malloc(sizeof(Stack)) ;
        initStack(stack) ;

        /** init the buffer */
        DBuffer * dbuffer = (DBuffer *)malloc(sizeof(DBuffer)) ;
        if(!initDBuffer(dbuffer,processedFileName))
        {
            return FALSE ;
        }
        /** init the result file*/
        char resultFileName[30] = "" ;
        char * dotPos = strchr(sourceFileName,'.') ;
        if(dotPos)
        {
            strncpy(resultFileName,sourceFileName,dotPos - sourceFileName) ;
            resultFileName[dotPos - sourceFileName] = '\0' ;
        }
        else
        {
            strcpy(resultFileName,sourceFileName) ;
        }
        strcat(resultFileName,".out") ;
        strcpy(outName,resultFileName) ;
        FILE * resultFile = fopen(resultFileName,"w") ;
        if(!resultFile)
        {
            printf("create output file %s failed !\n",resultFileName) ;
            return -1 ;
        }
        char c ;
        char tokenNameCopy[TOKEN_MAX_LENGTH] ;
```

```
        while((c = getChar(dbuffer)) != EOF)
        {
            if( c == ' ' || c == '\n' || c == '\r') //in fact , there are no '\r' any more
            {
                continue ;
            }
            /** recognize the ID , KEYWORD */
            else if(isNonedigit(c)) /* identifier = nonedigit(nonedigit|digit)* */
            {
                readyCopy(dbuffer) ;
                //read until not  (nonedigit|digit)
                c = getChar(dbuffer) ;
                while(isNonedigit(c) || isdigit(c) )
                {
                    c = getChar(dbuffer) ;
                }
                /* the identifier has over */
                retract(dbuffer,1) ; /* retract ,we should retract before copy it !!*/
                copyToken(dbuffer,tokenNameCopy) ;
                /* is the key word ?*/
                if(lookUpKeyWord(keyWord,tokenNameCopy))
                {
                    capitalize((char *)tokenNameCopy) ;
                    tokenScanEcho(tokenNameCopy,"") ;
                    tokenScanWriteToFile(resultFile,tokenNameCopy,"") ;
                    continue ;
                }
                /* not the keyword */
                tokenScanEcho("ID",tokenNameCopy) ;
                tokenScanWriteToFile(resultFile,"ID",tokenNameCopy) ;
            }
            /** recognize the INT ,REAL */
            else if(isdigit(c))
            {
                readyCopy(dbuffer) ;
                c = getChar(dbuffer) ;
                while(isdigit(c))
                {
                   c = getChar(dbuffer) ;
                }
                /* it is not the digit , is '.' ?*/
                if(c == '.')
                {
                    c = getChar(dbuffer) ;
```

```
                while(isdigit(c))
                {
                    c = getChar(dbuffer) ;
                }
                /* real over */
                retract(dbuffer,1) ;
                copyToken(dbuffer,tokenNameCopy) ;

                /* just echo */
                tokenScanEcho("REAL_C",tokenNameCopy) ;
                tokenScanWriteToFile(resultFile,"REAL_C",tokenNameCopy) ;
            }
            else
            {
                /* it is INT */
                retract(dbuffer,1) ;
                copyToken(dbuffer,tokenNameCopy) ;

                /* just echo */
                tokenScanEcho("INT_C",tokenNameCopy) ;
                tokenScanWriteToFile(resultFile,"INT_C",tokenNameCopy) ;
            }
        }
        /** recognize the CHAR_C */
        else if(c == '\'')
        {
            readyCopy(dbuffer) ;
            c = getChar(dbuffer) ;
            if(c == '\\')
            {
                /* escape char */
                c = getChar(dbuffer) ;
                /* again */
                c = getChar(dbuffer) ;
                if(c != '\'')
                {
                    tokenScanError("CHAR definition error","char escape error") ;
                }
                copyToken(dbuffer,tokenNameCopy) ;
                tokenScanEcho("CHAR_C",tokenNameCopy) ;
                tokenScanWriteToFile(resultFile,"CHAR_C",tokenNameCopy) ;
            }
            else
            {
```

```
                    c = getChar(dbuffer) ;
                    if(c != '\'')
                    {
                        tokenScanError("CHAR  definition  error","'should  just  has  one
character") ;
                        errc++ ;
                    }
                    copyToken(dbuffer,tokenNameCopy) ;
                    /* do not need retract */
                    tokenScanEcho("CHAR_C",tokenNameCopy) ;
                    tokenScanWriteToFile(resultFile,"CHAR_C",tokenNameCopy) ;
                }
            }
            /** recognize the string */
            else if(c == '"')
            {
                readyCopy(dbuffer) ;
                c = getChar(dbuffer) ;
                while(c != '"')
                {
                    if(c == '\\')
                    {
                        /* escape , be careful */
                        /* we read the char to skip the escape */
                        c = getChar(dbuffer) ;

                    }
                    c = getChar(dbuffer) ;
                }
                copyToken(dbuffer,tokenNameCopy) ;
                /* do not retract */
                tokenScanEcho("STRING_C",tokenNameCopy) ;
                tokenScanWriteToFile(resultFile,"STRING_C",tokenNameCopy) ;
            }
            /** recognize the operator and separator*/
            else
            {
                /*readyCopy(dbuffer) ;*/
                switch(c)
                {
                    case '>' :
                    {
                        c = getChar(dbuffer) ;
                        if(c == '=')
```

```
                    {
                        /** >= */
                        tokenScanEcho("GE","") ;
                        /*tokenScanEcho("=","") ;*/
                        tokenScanWriteToFile(resultFile,"GE","") ;
                    }
                    else
                    {
                        /** > */
                        retract(dbuffer,1) ;
                        tokenScanEcho("GT","") ;
                         tokenScanWriteToFile(resultFile,"GT","") ;
                    }
                    break ;
                }
                case '<' :
                {
                    c = getChar(dbuffer) ;
                    if(c == '=')
                    {
                        /** <= */
                        tokenScanEcho("LE","") ;
                        tokenScanWriteToFile(resultFile,"LE","") ;
                    }
                    else
                    {
                        /** < */
                        retract(dbuffer,1) ;
                        tokenScanEcho("LT","") ;
                        tokenScanWriteToFile(resultFile,"LT","") ;
                    }
                    break ;
                }
                case '=' :
                {
                    c = getChar(dbuffer) ;
                    if(c == '=')
                    {
                        /** == */
                        tokenScanEcho("EQ","") ;
                        tokenScanWriteToFile(resultFile,"EQ","") ;
                    }
                    else
                    {
```

```
                        /** = */
                        retract(dbuffer,1) ;
                        tokenScanEcho("ASSIGN","") ;
                        tokenScanWriteToFile(resultFile,"ASSIGN","") ;
                    }
                    break ;
                }
                case '!' :
                {
                    c = getChar(dbuffer) ;
                    if(c == '=')
                    {
                        /** != */
                        tokenScanEcho("NEQ","") ;
                        tokenScanWriteToFile(resultFile,"NEQ","") ;
                    }
                    else
                    {
                        /** ! */
                        retract(dbuffer,1) ;
                        tokenScanEcho("NOT","") ;
                        tokenScanWriteToFile(resultFile,"NOT","") ;
                    }
                    break ;
                }
                case '(' :
                {
                     /** ( */
                    tokenScanEcho("LR_BRAC","") ;
                    tokenScanWriteToFile(resultFile,"LR_BRAC","") ;
                    /* we need do the error-process by stack */
                    if(!push(stack,'('))
                    {
                        tokenScanError("INNER ERROR","stack has full") ;
                        errc++ ;
                    }
                    break ;
                }

                case ')' :
                {
                     /** ) */
                    tokenScanEcho("RR_BRAC","") ;
                    tokenScanWriteToFile(resultFile,"RR_BRAC","") ;
```

```
                        char charMatch ;
                        if(pop(stack,&charMatch))
                        {
                            if(charMatch != '(')
                            {
                                /* not matched */
                                tokenScanError("NOT  MATCHED","right  round  bracket  is  not
matched") ;
                                errc++ ;
                            }
                        }
                        else
                        {
                            /* the stack has empty*/
                            tokenScanError("NOT  MATCHED","more  right  round  brackets  is
found") ;
                            errc++ ;
                        }
                        break ;
                    }
                    case '[' :
                    {
                          /** [ */
                        tokenScanEcho("LS_BRAC","") ;
                        tokenScanWriteToFile(resultFile,"LS_BRAC","") ;
                        if(!push(stack,'['))
                        {
                            tokenScanError("INNER ERROR","stack has full") ;
                            errc++ ;
                        }
                        break ;
                    }

                    case ']' :
                    {
                        tokenScanEcho("RS_BRAC","") ;
                        tokenScanWriteToFile(resultFile,"RS_BRAC","") ;
                        char charMatch ;
                        if(pop(stack,&charMatch))
                        {
                            if(charMatch != '[')
                            {
                                /* not matched */
                                tokenScanError("NOT  MATCHED","right  square  bracket  is  not
```

```
matched") ;
                                errc++ ;
                            }
                        }
                        else
                        {
                            /* the stack has empty*/
                            tokenScanError("NOT  MATCHED","more  right  square  brackets  is
found") ;
                            errc++ ;
                        }
                        break ;
                    }
                    case '+':
                    {
                        c = getChar(dbuffer) ;
                        if(c == '+')
                        {
                            /** ++ */
                            tokenScanEcho("INC","") ;
                            tokenScanWriteToFile(resultFile,"INC","") ;
                        }
                        else if(c == '=')
                        {
                            /** ++ */
                            tokenScanEcho("ADD_ASS","") ;
                            tokenScanWriteToFile(resultFile,"ADD_ASS","") ;
                        }
                        else
                        {
                            /** + */
                            retract(dbuffer,1) ;
                            tokenScanEcho("ADD","") ;
                            tokenScanWriteToFile(resultFile,"ADD","") ;
                        }
                        break ;
                    }
                    case '-' :
                    {
                        c = getChar(dbuffer) ;
                        if(c == '-')
                        {
                            /** -- */
                            tokenScanEcho("DEC","") ;
```

```
                        tokenScanWriteToFile(resultFile,"DEC","") ;
                    }
                    else if(c == '=')
                    {
                        /** -= */
                        tokenScanEcho("SUB_ASS","") ;
                        tokenScanWriteToFile(resultFile,"SUB_ASS","") ;
                    }
                    else
                    {
                        /** - */
                        retract(dbuffer,1) ;
                        tokenScanEcho("SUB","") ;
                        tokenScanWriteToFile(resultFile,"SUB","") ;
                    }
                    break ;
                }
                case '*' :
                {
                    c = getChar(dbuffer) ;
                    if(c == '=')
                    {
                        /** *= */
                        tokenScanEcho("MUL_ASS","") ;
                        tokenScanWriteToFile(resultFile,"MUL_ASS","") ;
                    }
                    else
                    {
                        /** * */
                        retract(dbuffer,1) ;
                        tokenScanEcho("MUL_OR_INDIR","") ;
                        tokenScanWriteToFile(resultFile,"MUL_OR_INDIR","") ;
                    }
                    break ;
                }
                case '/' :
                {
                    c = getChar(dbuffer) ;
                    if(c == '=')
                    {
                        /** /= */
                        tokenScanEcho("DIV_ASS","") ;
                        tokenScanWriteToFile(resultFile,"DIV_ASS","") ;
                    }
```

```
                    else
                    {
                        /** / */
                        retract(dbuffer,1) ;
                        tokenScanEcho("DIV","") ;
                        tokenScanWriteToFile(resultFile,"DIV","") ;
                    }
                    break ;
                }
                case '&' :
                {
                    c = getChar(dbuffer) ;
                    if(c == '&')
                    {
                        /** && */
                        tokenScanEcho("AND","") ;
                        tokenScanWriteToFile(resultFile,"AND","") ;
                    }
                    else
                    {
                        /** &*/
                        retract(dbuffer,1) ;
                        tokenScanEcho("REFERENCE","") ;
                        tokenScanWriteToFile(resultFile,"REFERENCE","") ;
                    }
                    break ;
                }
                case '|' :
                {
                    c = getChar('|') ;
                    if(c == '|')
                    {
                        /** || */
                        tokenScanEcho("OR","") ;
                        tokenScanWriteToFile(resultFile,"OR","") ;
                    }
                    else
                    {
                        tokenScanError("NOT SUPPORTED OPARATOR","'|' is not supported!") ;
                        errc++ ;
                    }
                    break ;
                }
                case ',' :
```

```
                        /** , */
                        tokenScanEcho("COMMA","") ;
                        tokenScanWriteToFile(resultFile,"COMMA","") ;
                        break ;
                    case ';' :
                        /** ; */
                        tokenScanEcho("SEMIC","") ;
                        tokenScanWriteToFile(resultFile,"SEMIC","") ;
                        break ;
                    case '{' :
                    {
                        /** { */
                        tokenScanEcho("LB_BRAC","") ;
                        tokenScanWriteToFile(resultFile,"LB_BRAC","") ;
                        if(!push(stack,'{'))
                        {
                            errc++ ;
                            tokenScanError("INNER ERROR","stack has full") ;
                        }
                        break ;
                    }

                    case '}' :
                    {
                         /** } */
                        tokenScanEcho("RB_BRAC","") ;
                        tokenScanWriteToFile(resultFile,"RB_BRAC","") ;
                        char charMatch ;
                        if(pop(stack,&charMatch))
                        {
                            if(charMatch != '{')
                            {
                                /* not matched */
                                errc++ ;
                                tokenScanError("NOT MATCHED","right square bracket is not
matched") ;
                            }
                        }
                        else
                        {
                            /* the stack has empty*/
                            errc++ ;
                            tokenScanError("NOT MATCHED","more right square brackets is
found") ;
```

```
                    }
                    break ;
                }

                default :
                {
                    /** not supported */
                    char tokenError[200] ;
                    sprintf(tokenError,"character:%c is not supported!",c) ;
                    errc++ ;
                    tokenScanError("NOT SUPPORTED CHARACTER",tokenError) ;
                }

            }
        }
    }
    /* is the stack empty ? */
    if(!isEmpty(stack))
    {
        tokenScanError("NOT MATCHED","brackets is not matched") ;
    }
    deleteDBuffer(dbuffer) ;
    free(stack) ;
    fclose(resultFile) ;
    return errc ;
}
```

b) 语法分析

i. 分析表构建

```
int FIRST(int symbol , int set[] , int * setSize)
{
    if(getType(symbol) == TERMINAL)
    {
        addToSet(symbol,set,setSize) ;
        return EMPTY +1 ;
    }
    else
    {
        //find all productions
        int proNum = symbolList.data[symbol].productionIndex[0] ;
        int hasEmpty = EMPTY + 1 ;
        int k ;
        for(k = 0 ; k < proNum  ; k++)
        {
            int                                 len                                 =
```

```
productions.data[symbolList.data[symbol].productionIndex[k+1]].len ;
                if(len == 0)
                {
                    hasEmpty = EMPTY ;
                }
                else
                {
                    int                    *                    pRight                    =
productions.data[symbolList.data[symbol].productionIndex[k+1]].pRight ;
                    int j = 0 ;
                    while(j < len)
                    {
                        if(FIRST(pRight[j],set , setSize) != EMPTY)
                        {
                            break ;
                        }
                        else
                        {
                            j++ ;
                        }
                    }
                    if(j == len)
                    {
                        //means that all has lead to empty
                        hasEmpty = EMPTY ;
                    }
                }
            }
            return hasEmpty ;
        }
    }
    void getFirstCollection(int * p ,int pLen , int * firstSet ,int * fsLen,int * hasEmpty)
    {
        if(pLen == 0)
        {
            (*hasEmpty) = EMPTY ;
            return ;
        }
        else
        {
            if(getType(p[0]) == TERMINAL)
            {
                firstSet[0] = p[0] ;
                (*fsLen) = 1 ;
```

```
            (*hasEmpty) = 1 + EMPTY ;
        }
        else
        {
            int i = 0 ;
            while(i < pLen)
            {
                if(FIRST(p[i],firstSet,fsLen) != EMPTY)
                {
                    break ;
                }
                else
                {
                    i++ ;
                }
            }
            if(i == pLen)
            {
                (*hasEmpty) = EMPTY ;
            }
            else
            {
                (*hasEmpty) = EMPTY +1 ;
            }
        }
    }
}
int isLR1_ItemSame(LR1_Item * one , LR1_Item * two)
{
    if(one->productionOrder != two->productionOrder ||
           one->dotPos != two->dotPos ||
           one->exSymNum != two->exSymNum )
    {
        return SAME + 1 ;
    }
    else
    {
        int i ;
        for(i = 0 ; i < one->exSymNum ; i++)
        {
            int j ;
            int hasSame = SAME + 1 ;
            for(j = 0 ; j < two->exSymNum ; j++)
            {
```

```
                if(one->expectedSymbol[i] == two->expectedSymbol[j])
                {
                    hasSame = SAME ;
                    break ;
                }
            }
            if(hasSame != SAME)
            {
                return SAME +1 ;
            }
        }
        return SAME  ;
    }
}
int isStateSame(ItemCollection * one , ItemCollection * two)
{
    if(one->len != two->len)
    {
        return SAME + 1 ;
    }
    LR1_Item * oPos = one->items ;
    while(oPos != NULL)
    {
        LR1_Item * iPos = two->items ;
        while(iPos != NULL)
        {
            if(isLR1_ItemSame(iPos,oPos) == SAME)
            {
                break ;
            }
            iPos = iPos->next ;
        }
        if(iPos == NULL)
        {
            //it mean's no same as oPos
            return SAME +1 ;
        }
        oPos = oPos->next ;
    }
    return SAME ;
}
int CLOURE(ItemCollection * state)
{
#ifndef OUTPUT_LR1
```

```
        printf("I %d \n",state->order) ;
        LR1_Item * dPos = state->items ;
        while(dPos != NULL )
        {
            char * x = restoreP(dPos->productionOrder) ;
            printf("%s,%d,",x,dPos->dotPos) ;
            free(x) ;
            int di = 0 ;
            for( ; di < dPos->exSymNum ; di++)
            {
                printf("%s / ",getName(dPos->expectedSymbol[di])) ;
            }
            printf("\n") ;
            dPos = dPos->next ;
        }
    #endif
        int i ;
        LR1_Item * pos = state->items ;
        while(pos != NULL)
        {
            if(pos->dotPos == productions.data[pos->productionOrder].len)
            {
                /* redece item */
                state->hasReducedItem = TRUE_ANA ;
                pos = pos->next ;
            }
            else
            {
                int symbol = productions.data[pos->productionOrder].pRight[pos->dotPos] ;
                if(getType(symbol) == TERMINAL)
                {
                    /* no empty cloure */
                    pos = pos->next ;
                }
                else
                {
                    /* add the empty cloure of this LR1_Item to the end of the items chain */
                    int pNum = symbolList.data[symbol].productionIndex[0] ;
                    int j ;
                    /* first get the FIRST(Ba)*/
                    int firstSet[20] ;
                    int firstSetSize = 0 ;
                    int leftProduction[PRODUCTION_MAX_LEN] ;
                    int leftProductionLen = 0;
```

```
                    int hasEmpty ;
                    for(j = pos->dotPos + 1 ; j < productions.data[pos->productionOrder].len ;
j++)
                    {
                        leftProduction[leftProductionLen++]                                =
productions.data[pos->productionOrder].pRight[j] ;
                    }

getFirstCollection(leftProduction,leftProductionLen,firstSet,&firstSetSize,&hasEmpty) ;
                    if(hasEmpty == EMPTY)
                    {
                        //if has empty , add the current exSym to the new Item
                        for(j =0 ; j < pos->exSymNum ; j++)
                        {
                            addToSet(pos->expectedSymbol[j],firstSet,&firstSetSize) ;
                        }
                    }
                    //now can create the new Item
                    for(j = 0 ; j < pNum ; j++)
                    {
                        LR1_Item * newItem = (LR1_Item *)malloc(sizeof(LR1_Item)) ;
                        newItem->dotPos = 0 ;
                        newItem->productionOrder                                           =
productions.data[symbolList.data[symbol].productionIndex[j+1]].order ;
                        newItem->exSymNum = firstSetSize ;
                        int k ;
                        for(k = 0 ; k < firstSetSize ; k++)
                        {
                            newItem->expectedSymbol[k] = firstSet[k] ;
                        }
                        newItem->next = NULL ;
                        LR1_Item * testPos = state->items ;
                        while(testPos->next != NULL)
                        {
                            if(isLR1_ItemSame(testPos,newItem) == SAME)
                            {
                                free(newItem) ;
                                break ;
                            }
                            testPos = testPos->next ;
                        }
                        //because the testPos->next == NULL escape
                        if(testPos->next == NULL)
                        {
```

```
                        if(isLR1_ItemSame(testPos,newItem) != SAME)
                        {
                            testPos->next = newItem ;
#ifndef OUTPUT_LR1
                            char * x = restoreP(newItem->productionOrder) ;
                            printf("%s,%d,",x,newItem->dotPos) ;
                            int g = 0 ;
                            for( ; g < newItem->exSymNum ; g++)
                            {
                                printf("%s / ",getName(newItem->expectedSymbol[g])) ;
                            }
                            printf("\n") ;
                            free(x) ;
#endif
                            state->len++ ;
                        }
                        else
                        {
                            free(newItem) ;
                        }
                    }
                }
                pos = pos->next ;
            }
        }
    }
#ifndef OUTPUT_LR1
    printf("lenth is %d\n-----------\n",state->len) ;
#endif
    return 1 ;
}

int initItemCollection()
{
    LR1_Item * startItem = (LR1_Item*)malloc(sizeof(LR1_Item)) ;
    startItem->productionOrder = 0 ;
    startItem->dotPos = 0 ;
    startItem->exSymNum = 1 ;
    startItem->expectedSymbol[0] = transName("#") ;
    startItem->next = NULL ;
    ItemCollection * I0 = (ItemCollection * )malloc(sizeof(ItemCollection)) ;
    I0->items = startItem ;
    I0->len = 1 ;
    I0->order = 0 ;
```

```
        I0->hasReducedItem = FALSE_ANA ;
        ic.icNum = 0 ;
        ic.data[ic.icNum++] = I0 ;
        CLOURE(I0) ;
        /*ready the goList , reduceList */
        goHead = NULL ;
        goRear = NULL ;
        reduceHead = NULL ;
        reduceRear = NULL ;
        //create the successive item array
        struct SucItem
        {
            int readSymbol ;
            LR1_Item * sucItems ;
            int itemNum ;
        } ;
        struct SucItemList
        {
            struct SucItem data[SUCCESSIVE_ITEM_MAX_LEN] ;
            int siNum ;
        } si ;
        //
        int i ;
        for(i = 0 ; i < ic.icNum ; i++)
        {
            //first , init the successive Item array for current ItemCollection
            si.siNum = 0 ;
            /* for each itemCollection ,
                get successive item of each items in current itemCollection
                 if the item has successicve item
            */
            ItemCollection * current = ic.data[i] ;
            LR1_Item * pos = current->items ;
            while(pos != NULL)
            {
                if(pos->dotPos == productions.data[pos->productionOrder].len)
                {
                    /* redeced item*/
                    if(reduceHead != NULL)
                    {
                        struct ReduceList * newRL = (struct ReduceList *)malloc(sizeof(struct
ReduceList)) ;
                        newRL->next = NULL ;
                        newRL->state = i ;
```

```
                        newRL->reducedItem = pos ;
                        reduceRear->next = newRL ;
                        reduceRear = newRL ;
                    }
                    else
                    {
                        reduceHead = (struct ReduceList *)malloc(sizeof(struct ReduceList)) ;
                        reduceHead->state = current->order ;/* that is i */
                        reduceHead->reducedItem = pos ;
                        reduceHead->next = NULL ;
                        reduceRear = reduceHead ;
                    }
                    pos = pos->next ;
                }
                else
                {
                    /* it has successive item */
                    /* new a LR1_Item , which is this item's successive item*/
                    LR1_Item * newItem = (LR1_Item *)malloc(sizeof(LR1_Item)) ;
                    newItem->dotPos = pos->dotPos+1 ;
                    newItem->productionOrder = pos->productionOrder ;
                    newItem->exSymNum = pos->exSymNum ;
                    int j ;
                    for(j = 0 ; j < newItem->exSymNum ; j++)
                    {
                        newItem->expectedSymbol[j] = pos->expectedSymbol[j] ;
                    }
                    newItem->next = NULL ;
                    int                             readSymbol                             =
productions.data[pos->productionOrder].pRight[pos->dotPos] ;
                    int hasFound = FALSE_ANA ;
                    /* add the successive to the si */
                    for(j = 0 ; j < si.siNum ; j++)
                    {
                        if(readSymbol == si.data[j].readSymbol)
                        {
                            /* in si has the same symbol */
                            newItem->next = si.data[j].sucItems ;
                            si.data[j].sucItems = newItem ;
                            si.data[j].itemNum++ ;
                            hasFound = TRUE_ANA ;
                            break ;
                        }
                    }
```

```
                    if(hasFound == FALSE_ANA)
                    {
                        //si has not the symbol
                        //create one and add it
                        si.data[si.siNum].readSymbol = readSymbol ;
                        si.data[si.siNum].sucItems = newItem ;
                        si.data[si.siNum].itemNum = 1 ;
                        si.siNum++ ;
                    }
                    pos = pos->next ;
                }
            }
    #ifndef OUTPUT_SI
            int d_i ;
            printf("siNum = %d\n",si.siNum) ;
            for(d_i = 0 ; d_i < si.siNum ; d_i++)
            {
                printf("readSymbol:%s\n",getName(si.data[d_i].readSymbol)) ;
                LR1_Item * dPos = si.data[d_i].sucItems ;
                while(dPos != NULL)
                {

printf("----%s,exSymNum:%d\n",restoreP(dPos->productionOrder),dPos->exSymNum) ;
                    dPos = dPos->next ;
                }
            }
    #endif
            //and then , for each successive item , get is's empty cloure
            int x ;
            for(x = 0 ; x < si.siNum ; x++)
            {
                ItemCollection * newIC = (ItemCollection *) malloc(sizeof(ItemCollection)) ;
                newIC->len = si.data[x].itemNum ;
                newIC->items = si.data[x].sucItems ;
                newIC->order = ic.icNum ;
                //order has not init
                CLOURE(newIC) ;
                int k ;
                for(k = 0 ; k < ic.icNum ; k++)
                {
                    if(isStateSame(newIC,ic.data[k]) == SAME)
                    {
                        //do not add the IC , but goList should be update
                        if(goHead != NULL)
```

```
                        {
                            struct GOList * newGOL = (struct GOList *)malloc(sizeof(struct
GOList)) ;
                            newGOL->startState = i ;
                            newGOL->gotoState = k ;
                            newGOL->symbol = si.data[x].readSymbol ;
                            newGOL->next = NULL ;
                            goRear->next = newGOL ;
                            goRear = newGOL ;
                        }
                        else
                        {
                            goHead = (struct GOList *)malloc(sizeof(struct GOList)) ;
                            goHead->startState = i ;
                            goHead->gotoState = k ;
                            goHead->symbol = si.data[x].readSymbol ;
                            goHead->next = NULL ;
                            goRear = goHead ;
                        }
                        free(newIC) ;
                        break ;
                    }
                }
                if(k == ic.icNum)
                {
                    //mean newIC is new one ,add
                    ic.data[ic.icNum] = newIC ;
                    newIC->order = ic.icNum ;
                    // add this to the GOList
                    if(goHead != NULL)
                    {
                        struct GOList * newGOL = (struct GOList *)malloc(sizeof(struct
GOList)) ;
                        newGOL->startState = i ;
                        newGOL->gotoState = ic.icNum ;
                        newGOL->symbol = si.data[x].readSymbol ;
                        newGOL->next = NULL ;
                        goRear->next = newGOL ;
                        goRear = newGOL ;
                    }
                    else
                    {
                        goHead = (struct GOList *)malloc(sizeof(struct GOList)) ;
                        goHead->startState = i ;
```

```
                        goHead->gotoState = ic.icNum ;
                        goHead->symbol = si.data[x].readSymbol ;
                        goHead->next = NULL ;
                        goRear = goHead ;
                    }
                    ic.icNum++ ;
                }
            }
        }
    #ifndef OUTPUT_IC
        printf("\n%d",ic.icNum) ;
    #endif
    #ifndef OUTPUT_GOLIST
        struct GOList * gdpos = goHead ;
        while(gdpos != NULL)
        {

printf("%s,%d->%d\n",getName(gdpos->symbol),gdpos->startState,gdpos->gotoState) ;
            gdpos = gdpos->next ;
        }
    #endif
    #ifndef OUTPUT_REDUCE
        struct ReduceList * rdpos = reduceHead ;
        while(rdpos != NULL)
        {
            printf("%d,%s\n",rdpos->state,restoreP(rdpos->reducedItem->productionOrder));
            rdpos = rdpos->next ;
        }
    #endif
    }
    int initAnalysisTable()
    {
        if(initTransTableAndSymbolList() == -1)
        {
            return -1 ;
        }
        if(initProduction() == -1)
        {
            return -1 ;
        }
        initItemCollection() ;
        analysisTable.row = ic.icNum ;
        analysisTable.col = symbolList.symbolNum ;
        int k , l ;
```

```
        for(k = 0 ; k < analysisTable.row ; k++)
        {
            for(l = 0 ; l < analysisTable.col ; l++)
            {
                analysisTable.table[k][l] = ERROR ;
            }
        }
        /**
            if shift , the num is positive or 0 ,stands for the state to goto
            if reduce , the num is negative , stands for the production order which reduced
by
            if ACC , use MACRO ,which is expressed by a big positive num
            if error , use MACRO , which is expressed by a small negative num(abstract is large)
        */
        struct GOList * gpos = goHead ;
        while(gpos != NULL)
        {
            analysisTable.table[gpos->startState][gpos->symbol] = gpos->gotoState ;
            gpos = gpos->next ;
        }
        struct ReduceList * rpos = reduceHead ;
        while(rpos != NULL)
        {
            int i ;
            for(i = 0 ; i < rpos->reducedItem->exSymNum ; i++)
            {
                analysisTable.table[rpos->state][rpos->reducedItem->expectedSymbol[i]] = -
rpos->reducedItem->productionOrder ;
                if(rpos->reducedItem->productionOrder == 0)
                {
                    #ifdef SYNTAX
                    printf("find ACC\n") ;
                    #endif
                    analysisTable.table[rpos->state][rpos->reducedItem->expectedSymbol[i]]
= ACC ;
                }
            }
            rpos = rpos->next ;
        }
    #ifdef OUPUT_ANALYSISTABLE_FILE
        FILE * anaFile = fopen("analysisTable.txt","w") ;

        if(!anaFile)
        {
```

```
            printf("can not open file 'analysisTable.txt\n'") ;
            return -1 ;
        }
        char * x = " " ;
        fprintf(anaFile,"%10s",x) ;
        for(k = 0 ; k < analysisTable.col ; k++)
        {
            fprintf(anaFile,"%20s",getName(k)) ;
        }
        fprintf(anaFile,"\n") ;
        for(k = 0 ; k < analysisTable.row ; k++)
        {
            fprintf(anaFile,"%10d",k) ;
            for(l = 0 ; l < analysisTable.col ; l++)
            {
                int tVal = analysisTable.table[k][l] ;
                if(tVal == ERROR)
                {

                    fprintf(anaFile,"%20s",x) ;
                }
                else if(tVal >= 0 && tVal != ACC)
                {
                    fprintf(anaFile,"%20d",analysisTable.table[k][l]) ;
                }
                else if(tVal < 0 )
                {
                    char * s = restoreP(-tVal) ;
                    fprintf(anaFile,"%20s",s) ;
                    free(s) ;
                }
                else
                {
                    fprintf(anaFile,"ACC") ;
                }
            }
            fprintf(anaFile,"\n") ;
        }
    #endif
        return 0 ;
    }
```

ii. LR1 分析

```
    int syntaxAnalysis(char * fileName)
    {
```

```
        FILE * sourceFile = fopen(fileName,"r") ;
        if(!sourceFile)
        {
            printf("can not open file '%s'\n",fileName) ;
            return -1 ;
        }
        char  words[NAME_MAX_LEN] = "" ;
        char valOrAddr[VAL_MAX_LEN] = "" ;
        struct Stack
        {
            int data[STACK_DEPTH] ;
            int top ;
        } ; //EA stack
        struct Stack stateS ;
        struct Stack symbolS ;
        /* -----------init-------------- */
        stateS.top = 0 ;
        symbolS.top = 0 ;
        stateS.data[stateS.top++] = 0 ;
        symbolS.data[symbolS.top++] = transName("#") ;
            /** init semanticS */
        initSemanticS() ;
            /* make a global sybolTb */
        struct symbolTb * globalTb = mkTb(NULL) ;
            /* init the offset stack , tbptr stack */
        initTbptrS() ;
        initOffsetS() ;
        initLbptrS() ;
        offsetS.data[offsetS.top++] = 0 ;
        tbptrS.data[tbptrS.top++] = globalTb ;
        //-------------------
        #ifdef FREOPEN
        char * dotPos = strchr(fileName,'.') ;
        char intermediateFileName[30] ; //get the name
        if(dotPos == NULL)
        {
            strcpy(intermediateFileName,fileName) ;
            strcat(intermediateFileName,".ix") ;
        }
        else
        {
            char * dp = fileName ;
            for( ; dp != dotPos ; dp++)
            {
```

```
                intermediateFileName[dp - fileName] = *dp ;
            }
            intermediateFileName[dp - fileName] = '\0' ;
            strcat(intermediateFileName,".ix") ;
        }
        printf("%s",intermediateFileName) ;
        freopen(intermediateFileName , "w" , stdout) ; //to create or clear it
        freopen("CON","w",stdout) ;
        #endif
        getLineInfo(sourceFile,words,valOrAddr) ;
        while(1)
        {
            #ifdef OUTPUT_ANALYSIS_STACK
            int j = 0 ;
            for( ; j < stateS.top ; j++)
            {
                printf("%d,",stateS.data[j]) ;
            }
            printf("\n") ;
            for( j = 0 ; j < stateS.top ; j++)
            {
                printf("%s,",getName(symbolS.data[j]) ) ;
            }
            printf("\n") ;
            #endif
            #ifdef OUTPUT_READLINE
            printf("--words:%s\n--valoraddr:%s\n",words,valOrAddr) ;
            #endif
            #ifdef OUTPUT_SEMANTIC_STACK
            int dk = 0 ;
            printf("--semantic_stack--\n") ;
            for( ; dk < semanticS.top ; dk++)
            {
                printf("[%s,%s,%s]                                                                    |
",semanticS.data[dk].lexType,semanticS.data[dk].lexVal,semanticS.data[dk].tbName) ;
            }
            printf("\n") ;
            #endif
            int symbol = transName(words) ;
            if(symbol != EMPTY)
            {
                    // find ACTION
                    #ifdef OUTPUT_STATUS
                    int cState = stateS.data[stateS.top -1 ] ;
```

```
                    printf("current state=%d,symbol=%d,%s",cState,symbol,getName(symbol)) ;
                    #endif
                    int val = analysisTable.table[stateS.data[stateS.top -1 ]][symbol] ;
                    if(val == ACC )
                    {
                        #ifdef SYNTAX
                        printf("ACC\n") ;
                        #endif
                        return 0 ;
                    }
                    else if(val == ERROR)
                    {
                        #ifdef SYNTAX
                        printf("something error\n") ;
                        #endif
                        return -1 ;
                    }
                    else if(val >= 0 )
                    {
                        /* shift */
                        symbolS.data[symbolS.top++] = symbol ;
                        stateS.data[stateS.top++] = val ;
                        shiftSemanticAct(words,valOrAddr) ;
                        #ifdef SYNTAX
                        printf("shift %s\n",getName(symbol)) ;
                        #endif
                        /* read next */
                        if(getLineInfo(sourceFile,words,valOrAddr) == EOF)
                        {
                            strcpy(words,"#") ;
                        }
                    }
                    else if(val < 0)
                    {
                        /* reduce */
                        val = - val ;
                        /*print */

                        #ifdef SYNTAX
                        char * proS = restoreP(val) ;
                        printf("%s,%d\n",proS,strlen(proS)) ;
                        free(proS) ;
                        #endif
                        #ifdef FREOPEN
```

```
                        freopen(intermediateFileName,"a",stdout) ;
                        #endif

                        if(semanticAct(val) == FALSE_ANA)
                        {
                            #ifdef FREOPEN
                            fflush(stdout) ;
                            freopen( "CON", "w", stdout );
                            #endif
                            return FALSE_ANA ;
                        }



                        #ifdef FREOPEN
                        fflush(stdout) ;
                        freopen( "CON", "w", stdout );
                        #endif
                        //free(proS) ;
                        /* stack pop */
                        int len = productions.data[val].len ;
                        stateS.top = stateS.top - len ;
                        symbolS.top  = symbolS.top - len ;
                        /* stack push */
                            /* we should get the production left */
                        int pLeft = productions.data[val].pLeft ;
                        symbolS.data[symbolS.top++] = pLeft ;
                        /* look up goto table*/
                        int    gotoState    =    analysisTable.table[stateS.data[stateS.top
-1]][pLeft] ;
                        if(gotoState == ERROR)
                        {
                            printf("GOTO table occured error") ;
                            return -1 ;
                        }
                        stateS.data[stateS.top++] = gotoState ;
                    }
            }
            else
            {
                printf("not supported symbol ,%s!\n",words) ;
                return -1 ;
            }
        }
        return 0 ;
```

```
}
```

c) 语义分析

```
int semanticAct(int pdtIndex)
{
    char name[30] ;
    char type[30] ;
    int size ;
    void * addr ;
    struct symbolTb * curTb = tbptrS.data[tbptrS.top -1] ;
    char outputStr[300] ;
     switch(pdtIndex)
     {
          ...
          case ACT_paramList_type_ID_COMMA_paramList :
          {
           strcpy(name , semanticS.data[semanticS.top - 3].lexVal) ;
           strcpy(type , semanticS.data[semanticS.top - 4].lexVal) ;
           size = getSize(type) ;
           enterTb(curTb,name,type,offsetS.data[offsetS.top -1]) ;
           offsetS.data[offsetS.top -1]+= size ;
           //change semanticS
           semanticS.top -= 3 ;
           strcpy(semanticS.data[semanticS.top -1].lexType , "paramList") ;

               break ;
          }
          case ACT_paramList_type_ID :
          {
           //enter table
           strcpy(name , semanticS.data[semanticS.top -1].lexVal) ;
           strcpy(type,semanticS.data[semanticS.top -2].lexVal) ;
           size = getSize(type) ;
           addr = enterTb(curTb,name,type,offsetS.data[offsetS.top -1]) ;
           offsetS.data[offsetS.top -1]+=size ;
           //change semanticS
           semanticS.top -= 1 ;
           strcpy(semanticS.data[semanticS.top -1].lexType , "paramList") ;
               break ;
          }
          ...
          case ACT_idList_ID_COMMA_idList :
          {
              int k = 3 ;
           int isDeclare = 1 ;
```

```
                while(semanticS.top - k -1 >= 0)
                {
                    if((strcmp(semanticS.data[semanticS.top - k].lexType,"ID") == 0) &&
                        (strcmp(semanticS.data[semanticS.top - k -1].lexType,"type") == 0) )
                    {
                        //it is declare
                        break ;
                    }
                    else if((strcmp(semanticS.data[semanticS.top - k].lexType,"LR_BRAC") ==
0) &&
                        (strcmp(semanticS.data[semanticS.top - k -1].lexType,"ID") == 0))
                    {
                        //it is funApply
                        isDeclare = 0 ;
                        break ;
                    }
                    k++ ;
                }
                strcpy(name , semanticS.data[semanticS.top -3].lexVal) ;
                if(isDeclare)
                {
                    strcpy(type , semanticS.data[semanticS.top -k -1].lexVal) ;
                    #ifdef OUTPUT_TYPE
                    printf("----test_type:%s---\n",type) ;
                    #endif
                    size = getSize(type) ;
                    addr = enterTb(curTb , name , type , offsetS.data[offsetS.top -1]) ;
                    offsetS.data[offsetS.top -1]+=size ;
                    //change semanticS
                    semanticS.top -= 2 ;
                    strcpy(semanticS.data[semanticS.top -1].lexType , "idList") ;
                }
                else
                {
                    addr = lookupTb(curTb , name) ;
                    if(addr == NULL)
                    {
                        freopen("CON","w",stdout) ;
                        printf("\nvariable '%s' does not declared \n",name) ;
                        return FALSE_ANA ;
                    }
                    addr = ((struct tbNode *)addr)->addr ;
                    #ifdef ECHO_ASM
                    sprintf(outputStr,"param %s [push %p]\n",name,addr) ;
```

```
                    #else
                    sprintf(outputStr,"param %s\n",name) ;
                    #endif
                    //change semanticS
                    semanticS.top -= 2 ;
                    strcpy(semanticS.data[semanticS.top -1].lexType , "idList") ;
                    /* add the param num info*/
                    semanticS.data[semanticS.top -1].extend = semanticS.data[semanticS.top +
1].extend + 1;
                }
                     break ;
                }
                case ACT_idList_ID :
                {
                 //first to know is the declaration or the funApply
                 int k = 1 ;
                 int isDeclare = 1 ;
                 while(semanticS.top - k -1 >= 0)
                 {
                    if((strcmp(semanticS.data[semanticS.top - k].lexType,"ID") == 0) &&
                        (strcmp(semanticS.data[semanticS.top - k -1].lexType,"type") == 0) )
                    {
                        //it is declare
                        break ;
                    }
                    else if((strcmp(semanticS.data[semanticS.top - k].lexType,"LR_BRAC") ==
0) &&
                        (strcmp(semanticS.data[semanticS.top - k -1].lexType,"ID") == 0))
                    {
                        //it is funApply
                        isDeclare = 0 ;
                        break ;
                    }
                    k++ ;
                 }
                 strcpy(name , semanticS.data[semanticS.top -1].lexVal) ;
                 if(isDeclare)
                 {
                    strcpy(type , semanticS.data[semanticS.top -k -1].lexVal) ;
                    #ifdef OUTPUT_TYPE
                    printf("----test_type:%s---\n",type) ;
                    #endif
                    size = getSize(type) ;
                    addr = enterTb(curTb , name , type , offsetS.data[offsetS.top -1]) ;
```

```
                    offsetS.data[offsetS.top -1]+=size ;
                    //change semanticS
                    strcpy(semanticS.data[semanticS.top -1].lexType , "idList") ;
                }
                else
                {
                    addr = lookupTb(curTb , name) ;
                    if(addr == NULL)
                    {
                        freopen("CON","w",stdout) ;
                        printf("variable '%s' does not declared \n",name) ;
                        return FALSE_ANA ;
                    }
                    addr = ((struct tbNode *)addr)->addr ;
                    #ifdef ECHO_ASM
                    sprintf(outputStr,"param %s [push %p]\n",name,addr) ;
                    #else
                    sprintf(outputStr,"param %s\n",name) ;
                    #endif
                    genCode(outputStr) ;
                    strcpy(semanticS.data[semanticS.top -1].lexType , "idList") ;
                    /* add the param num info*/
                    semanticS.data[semanticS.top -1].extend = 1 ;
                }
                 break ;
            }
            case ACT_assignment_ID_ASSIGN_EXP_SEMIC :
            {
               //lookup the ID
             strcpy(name , semanticS.data[semanticS.top -4].lexVal) ;
             addr = lookupTb(curTb , name) ;
             if(addr == NULL)
             {
                freopen("CON","w",stdout) ;
                printf("variable '%s' has not been declared \n",name) ;
                return FALSE_ANA ;
             }
               //decide type
             char * tmpType = decideType(((struct tbNode *)addr)->type,
                                         (char *)semanticS.data[semanticS.top -2].extend) ;
             if(tmpType == NULL)
             {
                freopen("CON","w",stdout) ;
                printf("\nTYPE_CAST  ERROR  ,can  not  cast  '%s'  to  '%s'  \n",(char
```

```
*)semanticS.data[semanticS.top -2].extend,
                          ((struct tbNode *)addr)->type) ;
                    return FALSE_ANA ;
                }
                free(tmpType) ;
                #ifdef ECHO_ASM
                sprintf(outputStr , "%s = %s [mov ebx , %p ",name,semanticS.data[semanticS.top
- 2].lexVal,((struct tbNode *)addr)->addr) ;
                     genCode(outputStr) ;
                     addr = lookupTb(curTb , semanticS.data[semanticS.top -2].lexVal) ;
                     if(addr == NULL)
                     {
                         freopen("CON","w",stdout) ;
                    printf("\nvariable      '%s'      has      not      been      declared
\n",semanticS.data[semanticS.top -2].lexVal) ;
                    return FALSE_ANA ;
                     }
                     sprintf(outputStr,"                 mov  %p  ,ebx  ]",((struct  tbNode
*)addr)->addr) ;
                     #else
                     sprintf(outputStr,"%s  =  %s  \n",name,semanticS.data[semanticS.top  -
2].lexVal) ;
                     #endif
                     genCode(outputStr) ;
                //change semanticS
                semanticS.top -= 3 ;
                strcpy(semanticS.data[semanticS.top -1].lexType , "assignment") ;
                     break ;
               }
               case ACT_assignment_ID_ASSIGN_funApply :
               {

                //fist look this funApply has return val?
                if(strcmp(semanticS.data[semanticS.top -1].lexType,"fnRetVal") != 0)
                {
                    freopen("CON","w",stdout) ;
                    printf("\nfunction      '%s'      has      no      return      value
\n",semanticS.data[semanticS.top -1].lexVal) ;
                    return FALSE_ANA ;
                }
                strcpy(name , semanticS.data[semanticS.top -3].lexVal) ;
                addr = lookupTb(curTb , name) ;
                if(addr == NULL)
                {
```

```
                    printf("\nvariable '%s' has not been declared \n",name) ;
                    return FALSE_ANA ;
                }
                //decide type
                char * tmpType = decideType(((struct tbNode *)addr)->type ,
                                            (char *)semanticS.data[semanticS.top -1].extend) ;
                if(tmpType == NULL)
                {
                    freopen("CON","w",stdout) ;
                    printf("\nTYPE_CAST   ERROR   ,can   not   cast   %s   to   %s   \n",(char
*)semanticS.data[semanticS.top -1].extend,
                           (char *)semanticS.data[semanticS.top -3].extend) ;
                    return FALSE_ANA ;
                }
                free(tmpType) ;
                //genCode

                #ifdef ECHO_ASM
                sprintf(outputStr , "%s = %s [mov ebx , %p ",name,semanticS.data[semanticS.top
- 1].lexVal,((struct tbNode *)addr)->addr) ;
                     genCode(outputStr) ;
                     addr = lookupTb(curTb , semanticS.data[semanticS.top -1].lexVal) ;
                     if(addr == NULL)
                     {
                         freopen("CON","w",stdout) ;
                    printf("\nvariable      '%s'      has      not      been      declared
\n",semanticS.data[semanticS.top -2].lexVal) ;
                    return FALSE_ANA ;
                     }
                     sprintf(outputStr,"                mov  %p  ,ebx  ]",((struct  tbNode
*)addr)->addr) ;
                     #else
                     sprintf(outputStr,"%s  =  %s  \n",name,semanticS.data[semanticS.top  -
1].lexVal) ;
                     #endif
                     genCode(outputStr) ;
                     //change semanticS
                semanticS.top -= 2 ;
                strcpy(semanticS.data[semanticS.top -1].lexType , "assignment") ;
                     break ;
               }
               case ACT_assignment_singleOP_SEMIC :
               {
                   semanticS.top -- ;
```

```
                strcpy(semanticS.data[semanticS.top -1].lexType , "assignment") ;
                    break ;
                }
                case ACT_EXP_E_AND_EXP :
                {
                 char * tmpName = getTmpName() ;
                 strcpy(name ,tmpName) ;
                 free(tmpName) ;
                 strcpy(type,"BOOL") ;
                 size = getSize(type) ;
                 addr = enterTb(curTb,name,type,offsetS.data[offsetS.top-1]) ;
                 offsetS.data[offsetS.top -1]+=size ;
                 sprintf(outputStr,"%s  =  %s  and  %s  \n",name,semanticS.data[semanticS.top
-3].lexVal ,
                            semanticS.data[semanticS.top - 1].lexVal ) ;
                    genCode(outputStr) ;
                    //change semanticS
                    semanticS.top -= 2 ;
                    strcpy(semanticS.data[semanticS.top -1].lexVal,name) ;
                    strcpy((char *)semanticS.data[semanticS.top -1].extend , type) ;
                    break ;
                }
                case ACT_EXP_E_OR_EXP :
                {
                 char * tmpName = getTmpName() ;
                 strcpy(name ,tmpName) ;
                 free(tmpName) ;
                 strcpy(type,"BOOL") ;
                 size = getSize(type) ;
                 addr = enterTb(curTb,name,type,offsetS.data[offsetS.top-1]) ;
                 offsetS.data[offsetS.top -1]+=size ;
                 sprintf(outputStr,"%s  =  %s  or  %s  \n",name,semanticS.data[semanticS.top
-3].lexVal ,
                            semanticS.data[semanticS.top - 1].lexVal ) ;
                    genCode(outputStr) ;
                    //change semanticS
                    semanticS.top -= 2 ;
                    strcpy(semanticS.data[semanticS.top -1].lexVal,name) ;
                    strcpy((char *)semanticS.data[semanticS.top -1].extend , type) ;
                    break ;
                }
                case ACT_EXP_E :
                {
                 //nothing
```

```
                    break ;
                }
                case ACT_E_T_ADD_E :
                {
                 char * tmpName = getTmpName() ;
                 strcpy(name ,tmpName) ;
                 free(tmpName) ;
                 //decide the type ! at the op state , we should cast the small type to big type
                 //and the result type should be the big type
                 char * tmpType = decideType((char *)semanticS.data[semanticS.top -3].extend ,
                                             (char *)semanticS.data[semanticS.top -1].extend) ;
                 if(tmpType == NULL)
                 {
                     tmpType = decideType((char *)semanticS.data[semanticS.top -1].extend ,
                                          (char *)semanticS.data[semanticS.top -3].extend) ;
                 }
                 strcpy(type,tmpType) ;
                 free(tmpType) ;
                 size = getSize(type) ;
                 addr = enterTb(curTb,name,type,offsetS.data[offsetS.top-1]) ;
                 offsetS.data[offsetS.top -1]+=size ;
                 sprintf(outputStr,"%s  =  %s  +  %s  \n",name,semanticS.data[semanticS.top
-3].lexVal ,
                             semanticS.data[semanticS.top - 1].lexVal ) ;
                     genCode(outputStr) ;
                     //change semanticS
                     semanticS.top -= 2 ;
                     strcpy(semanticS.data[semanticS.top -1].lexVal,name) ;
                     strcpy((char *)semanticS.data[semanticS.top -1].extend , type) ;
                     break ;
                }
                ...
                case ACT_condition_IF_LR_BRAC_EXP_ACT_2_RR_BRAC_funBody :
                {
                 //genCode
                 sprintf(outputStr,"%s:\n",lbptrS.data[lbptrS.top -1]->data[1]) ;
                 genCode(outputStr) ;
                 //clear
                 free(lbptrS.data[lbptrS.top -1]) ;
                 lbptrS.top -- ;
                 //pop
                 curTb->width = offsetS.data[offsetS.top] ;
                 tbptrS.top-- ;
                 offsetS.top -- ;
```

```
                 //semanticS change
                 semanticS.top -= 5 ;
                 strcpy(semanticS.data[semanticS.top -1].lexType,"condition") ;
                      break ;
                }
                case ACT_condition_IF_LR_BRAC_EXP_ACT_2_RR_BRAC_funBody_ACT_3_ELSE_funBody :
                {
                 //genCode
                 sprintf(outputStr,"%s:\n",lbptrS.data[lbptrS.top -1]->data[2]) ;
                 genCode(outputStr) ;
                 //clear
                 free(lbptrS.data[lbptrS.top -1]) ;
                 lbptrS.top -- ;
                 //pop
                 curTb->width = offsetS.data[offsetS.top] ;
                 tbptrS.top-- ;
                 offsetS.top -- ;
                 //semanticS change
                 semanticS.top -= 8 ;
                 strcpy(semanticS.data[semanticS.top -1].lexType,"condition") ;
                      break ;
                }
                case
ACT_loop_FOR_LR_BRAC_forAssignPart_ACT_4_SEMIC_forBoolPart_ACT_5_SEMIC_forAssignPart_ACT_6
_RR_BRAC_funBody :
                {
                 //genCode
                 sprintf(outputStr,"goto %s\n%s:\n",lbptrS.data[lbptrS.top-1]->data[3],
                        lbptrS.data[lbptrS.top -1]->data[2]) ;
                 genCode(outputStr) ;
                 //clear
                 free(lbptrS.data[lbptrS.top -1]) ;
                 lbptrS.top -- ;
                 //pop
                 curTb->width = offsetS.data[offsetS.top] ;
                 tbptrS.top-- ;
                 offsetS.top -- ;
                 //semanticS change
                 semanticS.top -= 11 ;
                 strcpy(semanticS.data[semanticS.top -1].lexType,"condition") ;
                      break ;
                }
                case ACT_forAssignPart_forAssignList :
                {
```

```
                strcpy(semanticS.data[semanticS.top -1].lexType , "forAssignPart") ;
                     break ;
                }
                case ACT_forAssignPart :
                {
                 semanticS.top ++ ;
                 strcpy(semanticS.data[semanticS.top -1].lexType ,"forAssignPart") ;
                     break ;
                }
                case ACT_forAssignList_forAssignment_COMMA_forAssignList :
                {
                 semanticS.top -= 2 ;
                 strcpy(semanticS.data[semanticS.top -1].lexType ,"forAssignList") ;
                     break ;
                }
                case ACT_forAssignList_forAssignment :
                {
                 strcpy(semanticS.data[semanticS.top -1].lexType , "forAssignList") ;
                     break ;
                }
                case ACT_forAssignment_ID_ASSIGN_EXP :
                {
                 strcpy(name , semanticS.data[semanticS.top -3].lexVal) ;
                 addr = lookupTb(curTb , name) ;
                 if(addr == NULL)
                 {
                     freopen("CON","w",stdout) ;
                     printf("\nvariable '%s' has not been declared \n",name) ;
                     return FALSE_ANA ;
                 }
                 //decide type
                 char * tmpType = decideType(((struct tbNode *)addr)->type,
                                             (char *)semanticS.data[semanticS.top -1].extend) ;
                 if(tmpType == NULL)
                 {
                     freopen("CON","w",stdout) ;
                     printf("\nTYPE_CAST   ERROR   ,can   not   cast   %s   to   %s   \n",(char
*)semanticS.data[semanticS.top -1].extend,
                             (char *)semanticS.data[semanticS.top -3].extend) ;
                     return FALSE_ANA ;
                 }
                 free(tmpType) ;
                 #ifdef ECHO_ASM
                 sprintf(outputStr , "%s = %s [mov ebx , %p ",name,semanticS.data[semanticS.top
```

```
- 1].lexVal,((struct tbNode *)addr)->addr) ;
                    genCode(outputStr) ;
                    addr = lookupTb(curTb , semanticS.data[semanticS.top -1].lexVal) ;
                    if(addr == NULL)
                    {
                        freopen("CON","w",stdout) ;
                    printf("\nvariable      '%s'      has      not      been      declared
\n",semanticS.data[semanticS.top -2].lexVal) ;
                    return FALSE_ANA ;
                    }
                    sprintf(outputStr,"                 mov  %p  ,ebx  ]",((struct  tbNode
*)addr)->addr) ;
                    #else
                    sprintf(outputStr,"%s  =  %s  \n",name,semanticS.data[semanticS.top  -
1].lexVal) ;
                    #endif
                    genCode(outputStr) ;
                //change semanticS
                semanticS.top -= 2 ;
                strcpy(semanticS.data[semanticS.top -1].lexType , "assignment") ;
                    break ;
               }

               case ACT_jumpWord_BREAK_SEMIC :
               {
                //find the FOR LABEL , if there is not , ERROR
                int k = semanticS.top - 3 ;
                while( k >= 0)
                {
                    if(strcmp(semanticS.data[k].lexType,"FOR") == 0)
                    {
                        break ;
                    }
                    k-- ;
                }
                if(k < 0 )
                {
                    freopen("CON","w",stdout) ;
                    printf("\nbreak Error ! loop not find \n") ;
                    return FALSE_ANA ;
                }
                else
                {
                    char * tmp = (char *)semanticS.data[k].extend ;
```

```
                    tmp = tmp + strlen(tmp)+1 ;
                    sprintf(outputStr,"goto %s\n",tmp) ;
                    genCode(outputStr) ;
                }
                //semanticS change
                semanticS.top -- ;
                strcpy(semanticS.data[semanticS.top -1].lexType,"jumpWord") ;
                     break ;
               }
               case ACT_returnWord_RETURN_returnVal_SEMIC :
               {
                strcpy(name ,semanticS.data[semanticS.top -2].lexVal) ;
                addr = lookupTb(curTb,name) ;
                if(addr == NULL)
                {
                    freopen("CON","w",stdout) ;
                    printf("\nvariable '%s' has not been declared \n",name) ;
                    return FALSE_ANA ;
                }
                addr = ((struct tbNode *)addr)->addr ;
                #ifdef ECHO_ASM
                sprintf(outputStr , "eax = %s [mov eax , %p]\n",name , addr) ;
                #else
                sprintf(outputStr ,"eax = %s \n",name) ;
                #endif
                genCode(outputStr) ;
                //echo goto label
                if(lbptrS.top <= 0 )
                {
                    freopen("CON","w",stdout) ;
                    printf("lbptrS has been in mess\n") ;
                    return FALSE_ANA ;
                }
                sprintf(outputStr,"goto %s\n",lbptrS.data[0]->data[0]) ;//the function's end
all have been save at here
                     genCode(outputStr) ;
                     //change semanticS
                     semanticS.top -= 2 ;
                     strcpy(semanticS.data[semanticS.top -1].lexType,"returnWord") ;
                     break ;
               }
               case ACT_returnVal_EXP :
               {
                strcpy(semanticS.data[semanticS.top -1].lexType,"returnVal") ;
```

```
                    break ;
                }
                case ACT_returnVal :
                {
                 semanticS.top ++ ;
                 strcpy(semanticS.data[semanticS.top -1].lexType,"returnVal") ;
                    break ;
                }
                case ACT_io_PRINTF_LR_BRAC_printContent_RR_BRAC_SEMIC :
                {
                 sprintf(outputStr,"call  printf  ,  %d\n",(int)semanticS.data[semanticS.top
-3].extend) ;
                    genCode(outputStr) ;
                    //change semanticS
                    semanticS.top -= 4 ;
                    strcpy(semanticS.data[semanticS.top -1].lexType ,"io") ;

                    break ;
                }
                case ACT_io_SCANF_LR_BRAC_STRING_C_COMMA_REFERENCE_ID_RR_BRAC_SEMIC :
                {
                 sprintf(outputStr,"param        %s\n",(char*)(semanticS.data[semanticS.top
-6].extend)) ;
                    genCode(outputStr) ;
                    free(semanticS.data[semanticS.top -6].extend) ;
                    addr = lookupTb(curTb,semanticS.data[semanticS.top -3].lexVal) ;
                    if(addr == NULL)
                    {
                        freopen("CON","w",stdout) ;
                        printf("\nvariable     '%s'     has     not     been     declared
\n",semanticS.data[semanticS.top -3].lexVal) ;
                        return FALSE_ANA ;
                    }
                    addr = ((struct tbNode *)addr)->addr ;
                    sprintf(outputStr,"param %p\n",addr) ;
                    genCode(outputStr) ;
                 sprintf(outputStr,"call scanf , 2\n") ;
                 genCode(outputStr) ;
                 //change semanticS
                 semanticS.top -= 7 ;
                 strcpy(semanticS.data[semanticS.top -1].lexType,"io") ;
                    break ;
                }
                case ACT_printContent_STRING_C :
```

```
               {
                sprintf(outputStr,"param   %s   \n",(char   *)semanticS.data[semanticS.top
-1].extend) ;
                genCode(outputStr) ;
                //chage semanticS
                free(semanticS.data[semanticS.top -1].extend) ;
                strcpy(semanticS.data[semanticS.top -1].lexType ,"printContent") ;
                //recode the paramNum
                semanticS.data[semanticS.top -1].extend = 1 ;
                     break ;
               }
               case ACT_printContent_STRING_C_COMMA_ID :
               {
                   //output
                   strcpy(name , semanticS.data[semanticS.top -1].lexVal) ;
                addr = lookupTb(curTb,name) ;
                if(addr == NULL)
                {
                    freopen("CON","w",stdout) ;
                    printf("\nvariable '%s' has not been declared\n",name) ;
                    return FALSE_ANA ;
                }
                #ifdef ECHO_ASM
                sprintf(outputStr,"param   %s   [push   %p]\n",name   ,   ((struct   tbNode
*)addr)->addr) ;
                #else
                sprintf(outputStr,"param %s \n",name) ;
                #endif
                genCode(outputStr) ;
                sprintf(outputStr,"param   %s   \n",(char   *)semanticS.data[semanticS.top
-3].extend) ;
                genCode(outputStr) ;
                free(semanticS.data[semanticS.top - 3].extend) ;
                //chage semanticS
                semanticS.top -= 2 ;
                strcpy(semanticS.data[semanticS.top -1].lexType ,"printContent") ;
                //recode the paramNum
                semanticS.data[semanticS.top -1].extend = 2 ;
                     break ;
               }
               case ACT_mainFun_INT_MAIN_ACT_1_LR_BRAC_formalParam_RR_BRAC_funBody :
               {
                //over the tb , offset
                tbptrS.data[tbptrS.top -1]->width = (int)offsetS.data[offsetS.top -1] ;
```

```
                tbptrS.top -- ;
                offsetS.top -- ;
                //output label
                sprintf(outputStr,"%s:\n",lbptrS.data[0]->data[0]) ;
                genCode(outputStr) ;
                //clear the lbptrS
                int k ;
                for(k = 0 ; k < lbptrS.top ; k++)
                {
                    free(lbptrS.data[k]) ;
                }
                lbptrS.top = 0 ;
                //change semanticS
                semanticS.top -= 6;
                strcpy(semanticS.data[semanticS.top -1].lexType , "mainFun") ;
                     break ;
               }
               case ACT_ACT_1 :
               {
                //create label
                struct LbS * glLabel = getNewLabelS() ;
                lbptrS.data[lbptrS.top ++] = glLabel ;
                createLabel(glLabel,1) ;
                //create symbol table , just enter the ID with type = TABLE
                //first , is it a fundefine or the MAIN , but it does not affect
                strcpy(name , semanticS.data[semanticS.top -1].lexVal) ;
                strcpy(type,"TABLE") ;
                strcpy((char  *)type  +  strlen(type)  +  1,semanticS.data[semanticS.top
-2].lexVal) ;
                addr = enterTb(curTb,name,type,offsetS.data[offsetS.top -1]) ;
                //push stack
                tbptrS.data[tbptrS.top++] = (struct symbolTb *)addr ;
                offsetS.data[offsetS.top++] = 0 ;
                /*out put label */
                sprintf(outputStr,"%s:\n",name) ;
                genCode(outputStr) ;
                //change semanticS
                semanticS.top ++ ;
                strcpy(semanticS.data[semanticS.top - 1].lexType ,"ACT_1") ;
                     break ;
               }
               case ACT_ACT_2 :
               {
                struct LbS * newLsS = getNewLabelS() ;
```

```
                createLabel(newLsS,2) ;
                lbptrS.data[lbptrS.top++] = newLsS ;
                sprintf(outputStr,"if               %s               goto               %s
\ngoto %s\n%s:\n",semanticS.data[semanticS.top-1].lexVal,
                           lbptrS.data[lbptrS.top-1]->data[0],lbptrS.data[lbptrS.top
-1]->data[1],
                           lbptrS.data[lbptrS.top-1]->data[0]) ;
                genCode(outputStr) ;
                //new symbol table
                char * tmpName = getTmpName() ;
                strcpy(type,"TABLE") ;
                strcpy((char *)type + strlen("TABLE") +1 , "IF") ;
                addr = enterTb(curTb,tmpName,type,offsetS.data[offsetS.top -1]) ;
                //change table
                tbptrS.data[tbptrS.top++] = (struct symbolTb *)addr ;
                offsetS.data[offsetS.top++] = 0 ;
                //change semantic
                semanticS.top ++ ;
                strcpy(semanticS.data[semanticS.top -1].lexType , "ACT_2") ;
                    break ;
               }
               case ACT_ACT_3 :
               {
                createLabel(lbptrS.data[lbptrS.top -1],1) ;
                sprintf(outputStr ,"goto %s\n%s\n",lbptrS.data[lbptrS.top-1]->data[2],
                       lbptrS.data[lbptrS.top -1]->data[1]) ;
                genCode(outputStr) ;
                //finish if's symbol table
                curTb->width = (int)offsetS.data[offsetS.top -1] ;
                tbptrS.top -- ;
                offsetS.top -- ;
                curTb = tbptrS.data[tbptrS.top -1] ;
                //new symbolTable
                char * tmpName = getTmpName() ;
                strcpy(type,"TABLE") ;
                strcpy((char *)type + strlen("TABLE") +1 , "ELSE") ;
                addr = enterTb(curTb,tmpName,type,offsetS.data[offsetS.top -1]) ;
                //change table
                tbptrS.data[tbptrS.top++] = (struct symbolTb *)addr ;
                offsetS.data[offsetS.top++] = 0 ;
                //change semantic
                semanticS.top ++ ;
                strcpy(semanticS.data[semanticS.top -1].lexType , "ACT_3") ;
                    break ;
```

```
                }
                case ACT_ACT_4 :
                {
                 //create label
                 struct LbS * newLabel = getNewLabelS() ;
                 createLabel(newLabel,4) ;
                 lbptrS.data[lbptrS.top++] = newLabel ;
                 sprintf(outputStr,"%s:\n",lbptrS.data[lbptrS.top -1]->data[0]) ;
                 genCode(outputStr) ;
                 semanticS.top ++ ;
                 strcpy(semanticS.data[semanticS.top -1].lexType , "ACT_4") ;
                 /* notice ! because the 'break' and 'continue' should know the
                  label of the loop angin and loop over , and the break and continue
                  can be at anywhere ,so it is hard to find if we just store the
                  label at the lbptrS ,so we store this two label at FOR block at
                  semanticS  */
                  /* find the FOR block */
                  int ForPos = semanticS.top - 4 ;
                  if(strcmp(semanticS.data[ForPos].lexType,"FOR") != 0)
                  {
                      freopen("CON","w",stdout) ;
                      printf("The semantic stack has been in mess !\n") ;
                      return FALSE_ANA ;
                  }
                  char * labelRecord = (char *)malloc(sizeof(char)*10*2) ;
                  strcpy(labelRecord ,lbptrS.data[lbptrS.top -1]->data[3]) ;//get continue
label
                  strcpy(labelRecord  +  strlen(labelRecord)  +  1,lbptrS.data[lbptrS.top
-1]->data[2]) ;//break label
                   semanticS.data[ForPos].extend = (void *)labelRecord ;
                      break ;
                }
                case ACT_ACT_5 :
                {
                 //genCode
                 sprintf(outputStr,"if              %s              goto              %s
\ngoto %s\n%s:\n",semanticS.data[semanticS.top -1].lexVal,
                        lbptrS.data[lbptrS.top -1]->data[1] ,
                        lbptrS.data[lbptrS.top -1]->data[2] ,
                        lbptrS.data[lbptrS.top -1]->data[3]) ;
                 genCode(outputStr) ;
                 semanticS.top ++ ;
                 strcpy(semanticS.data[semanticS.top -1].lexType , "ACT_5") ;
                      break ;
```

```
            }
            case ACT_ACT_6 :
            {
             sprintf(outputStr,"goto %s\n%s:\n",lbptrS.data[lbptrS.top -1]->data[0],
                              lbptrS.data[lbptrS.top -1]->data[1]) ;
             genCode(outputStr) ;
             //create symbolTb
             char * tmpName = getTmpName() ;
             strcpy(type,"TABLE") ;
             strcpy((char *)type + strlen("TABLE") +1 , "FOR") ;
             addr = enterTb(curTb,tmpName,type,offsetS.data[offsetS.top -1]) ;
             //change table
             tbptrS.data[tbptrS.top++] = (struct symbolTb *)addr ;
             offsetS.data[offsetS.top++] = 0 ;
             //change semantic
             semanticS.top ++ ;
             strcpy(semanticS.data[semanticS.top -1].lexType , "ACT_6") ;
                 break ;
            }
            default:
             freopen("CON","w",stdout) ;
             printf("\nERROR!\n") ;
                 break ;
        }
        return TRUE_ANA ;
    }
```

测试

1. 测试样例

```
    void myPutC(char c)
    {
        printf("%c",c) ;
    }

    int main()
    {
        //test declaration
        int a , b ;
        //test assign
        a = 4 ;
        b = 3 ;
        int k ;
        k = a - b ;
```

```
        bool xxb ;
        xxb = a == 4 ;
        //test if
        if(k > 0 && xxb)
        {
            //test printf
            printf("k'val is %s\n",k) ;
        }
        else
        {
            printf("k is less than 0\n") ;
        }
        //test loop
        int i  ;
        char xxc ;
        for(i = 3 , xxc = 'x' ; i > 0 ; i--)
        {
            if(i == k)
            {
                double dnum ;
                dnum = a ;
                //b = dbum ; //cast error
                //test break
                break ;
            }
            else
            {
                xxc = 'e' ;
                myPutC(xxc) ;
                //test continue ;
                continue ;
            }
        }
        //test return
        return 0 ;
    }
```

2. 输出结果

```
    myPutC:
    param c
    param "%c"
    call printf , 2
    label_0:
    MAIN:
    tmp_0 = 4
```

```
a = tmp_0
tmp_1 = 3
b = tmp_1
tmp_2 = a - b
k = tmp_2
tmp_3 = 4
tmp_4 = a == tmp_3
xxb = tmp_4
tmp_5 = 0
tmp_6 = k > tmp_5
tmp_7 = tmp_6 and xxb
if tmp_7 goto label_2
goto label_3
label_2:
param k
param "k'val is %s\n"
call printf , 2
goto label_4
label_3
param "k is less than 0\n"
call printf , 1
label_4:
tmp_10 = 3
i = tmp_10
tmp_11 = 'x'
xxc = tmp_11
label_5:
tmp_12 = 0
tmp_13 = i > tmp_12
if tmp_13 goto label_6
goto label_7
label_8:
DEC i
goto label_5
label_6:
tmp_15 = i == k
if tmp_15 goto label_9
goto label_10
label_9:
dnum = a
goto label_7
goto label_11
label_10
tmp_18 = 'e'
```

```
xxc = tmp_18
param xxc
call myPutC , 1
goto label_8
label_11:
goto label_8
label_7:
tmp_19 = 0
eax = tmp_19
goto label_1
label_1:
```

总结

通过构建简单编译器的前端，加深了对编译原理课本知识的理解，提高了将理论算法转换为实际代码的能力，也认识到了理论和实际上的差距。

印象最深刻的就是构造 LR1 分析表的过程，当初选择手工构造，只是觉得从编译工作台的输出中导入进来同样很麻烦，同时看到编译工作台这么牛的软件，不仅能生成各种分析表，还可以做动态分析，大受振奋，希望自己也至少写个 LR1 分析表的构建函数来。这大概花了一周的所有空闲时间。其中遇到了很多问题。最让人觉得神奇的就是，我所写的 LR1 分析表是不能分析含有左递归的产生式的，但是编译工作台却可以。我猜想如果不是采用其他的算法的话，那么他应该是在求空闭包的时候设置了递归层数的限制，这样当遇到左递归产生式而出现无穷递归时，当递归层次到了临界值，则不再递归，而是返回，跳到下一个产生式。这只是一个猜测，没有具体的实现。最后还是佩服下编译工作台的作者，实在是很厉害。

在做语义分析的时候，开始是一直在纠结怎么实现自底向上的值传递，即 L 属性和 S 属性的翻译问题。看了大概有两天，终于从书上的一小节找到了答案，一切都是通过一个语义栈来实现的。有了这个语义栈，一切信息都可以访问得到，而且整个语义分析就是通过操纵这个栈完成的。书上讲述的 S 属性和 L 属性的翻译，只是理论上的探讨，对于实际实现其实没有很大的意义。