代码说明文档

1 词法分析器

1.1 说明

词法分析器接受一个正规文法产生式文件和一个源代码文件，根据此文法产生式判断源代码文件中的内容是否符合此文法，识别并生成一个 token 表。表项形式为(所在行号，内容，token 类别)。token 类别分为五大类：关键词，标识符，常量（包括字符串、字符、复数、科学计数），限定符，运算符。

1.2 输入

1.2.1 正规文法文件：正规文法产生式.txt

K 表示非终结符集V_N

S 表示开始符 S

sigma 表示终结符集V_T

＜非终结符＞ → ＜终结符＞＜非终结符＞表示规则 P

ε直接忽略即可

```
K:SABCDFGHIJKLMNOPQRSTUVE

S:S
sigma:0123456789abcdefghijklmnopqrstuvwxyz.
+_ABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&*()-=/:;"'{}[]|\,<>?~`
N->
M->
O->
B->
U->
E->
S->0B
S->0D
S->1A
S->2A
```

```
S->3A
S->4A
S->5A
S->6A
S->7A
S->8A
S->9A
A->0A
A->1A
A->2A
A->3A
A->4A
A->5A
A->6A
A->7A
A->8A
A->9A
A->B
B->.C
C->A
D->.F
F->0F
F->1F
F->2F
F->3F
F->4F
F->5F
F->6F
F->7F
F->8F
F->9F
F->EG
G->+H
H->0O
H->1I
H->2I
H->3I
H->4I
H->5I
H->6I
H->7I
H->8I
H->9I
I->0I
```

```
I->1I
I->2I
I->3I
I->4I
I->5I
I->6I
I->7I
I->8I
I->9I
I->O
B->+J
J->1K
J->2K
J->3K
J->4K
J->5K
J->6K
J->7K
J->8K
J->9K
K->0K
K->1K
K->2K
K->3K
K->4K
K->5K
K->6K
K->7K
K->8K
K->9K
K->L
J->0L
L->iM
S->aN
S->bN
S->cN
S->dN
S->eN
S->fN
S->gN
S->hN
S->iN
S->jN
S->kN
```

```
S->lN
S->mN
S->nN
S->oN
S->pN
S->qN
S->rN
S->sN
S->tN
S->uN
S->vN
S->wN
S->xN
S->yN
S->zN
S->_N
S->AN
S->BN
S->CN
S->DN
S->EN
S->FN
S->GN
S->HN
S->IN
S->JN
S->KN
S->LN
S->MN
S->NN
S->ON
S->PN
S->QN
S->RN
S->SN
S->TN
S->UN
S->VN
S->WN
S->XN
S->YN
S->ZN
N->aN
N->bN
```

```
N->cN
N->dN
N->eN
N->fN
N->gN
N->hN
N->iN
N->jN
N->kN
N->lN
N->mN
N->nN
N->oN
N->pN
N->qN
N->rN
N->sN
N->tN
N->uN
N->vN
N->wN
N->xN
N->yN
N->zN
N->AN
N->BN
N->CN
N->DN
N->EN
N->FN
N->GN
N->HN
N->IN
N->JN
N->KN
N->LN
N->MN
N->NN
N->ON
N->PN
N->QN
N->RN
N->SN
N->TN
```

```
N->UN
N->VN
N->WN
N->XN
N->YN
N->ZN
N->0N
N->1N
N->2N
N->3N
N->4N
N->5N
N->6N
N->7N
N->8N
N->9N
N->_N
S->"P
P->aP
P->bP
P->cP
P->dP
P->eP
P->fP
P->gP
P->hP
P->iP
P->jP
P->kP
P->lP
P->mP
P->nP
P->oP
P->pP
P->qP
P->rP
P->sP
P->tP
P->uP
P->vP
P->wP
P->xP
P->yP
P->zP
```

```
P->AP
P->BP
P->CP
P->DP
P->EP
P->FP
P->GP
P->HP
P->IP
P->JP
P->KP
P->LP
P->MP
P->NP
P->OP
P->PP
P->QP
P->RP
P->SP
P->TP
P->UP
P->VP
P->WP
P->XP
P->YP
P->ZP
P->0P
P->1P
P->2P
P->3P
P->4P
P->5P
P->6P
P->7P
P->8P
P->9P
P->:P
P->;P
P-> P
P->/P
P->[P
P->]P
P->{P
P->}P
```

```
P->|P
P->!P
P->@P
P->#P
P->$P
P->%P
P->^P
P->&P
P->*P
P->(P
P->)P
P->-P
P->+P
P->_P
P->=P
P->.P
P->~P
P->`P
P-><P
P->>P
P->,P
P->?P
R->aT
R->bT
R->cT
R->dT
R->eT
R->fT
R->gT
R->hT
R->iT
R->jT
R->kT
R->lT
R->mT
R->nT
R->oT
R->pT
R->qT
R->rT
R->sT
R->tT
R->uT
R->vT
```

```
R->wT
R->xT
R->yT
R->zT
R->AT
R->BT
R->CT
R->DT
R->ET
R->FT
R->GT
R->HT
R->IT
R->JT
R->KT
R->LT
R->MT
R->NT
R->OT
R->PT
R->QT
R->RT
R->ST
R->TT
R->UT
R->VT
R->WT
R->XT
R->YT
R->ZT
R->0T
R->1T
R->2T
R->3T
R->4T
R->5T
R->6T
R->7T
R->8T
R->9T
R->:T
R->;T
R->/T
R->[T
```

```
R->]T
R->{T
R->}T
R->|T
R->!T
R->@T
R->#T
R->$T
R->%T
R->^T
R->&T
R->*T
R-> T
R->(T
R->)T
R->-T
R->+T
R->_T
R->=T
R->.T
R->~T
R->`T
R-><T
R->>T
R->,T
R->?T
P->\Q
Q->'P
Q->"P
P->"E
S->'R
R->\V
V->\T
V->'T
V->"T
T->'U
```

1.2.2 源代码文件：demo.txt

支持简单 C++语法，扩充了复数（形如：1+5i）和科学计数法（形如：0.123E+5）

```
void myPrint(int i,int j){
    return i+j;
}
int main(){
```

```
    int n = 5,m = 3;
    int pp[3]={1,2,3};
    int*p=&n;
    bool isOK = true;
    n+=(3*m;
    myPrint(n,m)
    double xs = 156.154;
    double d=0.123E+5;
    double fs=( 5+9i )+5;
    string s = "hello \"world!";
    char c = s[1;
    if(c == 'a'&&(n==m || isOK)){
        n++;
    }
    return 0;
```

1.3 输出

token 表文件：Token.txt

```
1 关键词 void
1 标识符 myPrint
1 限定符 (
1 关键词 int
1 标识符 i
1 限定符 ,
1 关键词 int
1 标识符 j
1 限定符 )
1 限定符 {
2 关键词 return
2 标识符 i
2 运算符 +
2 标识符 j
2 限定符 ;
3 限定符 }
4 关键词 int
4 标识符 main
4 限定符 (
4 限定符 )
4 限定符 {
5 关键词 int
5 标识符 n
5 运算符 =
5 常量 5
```

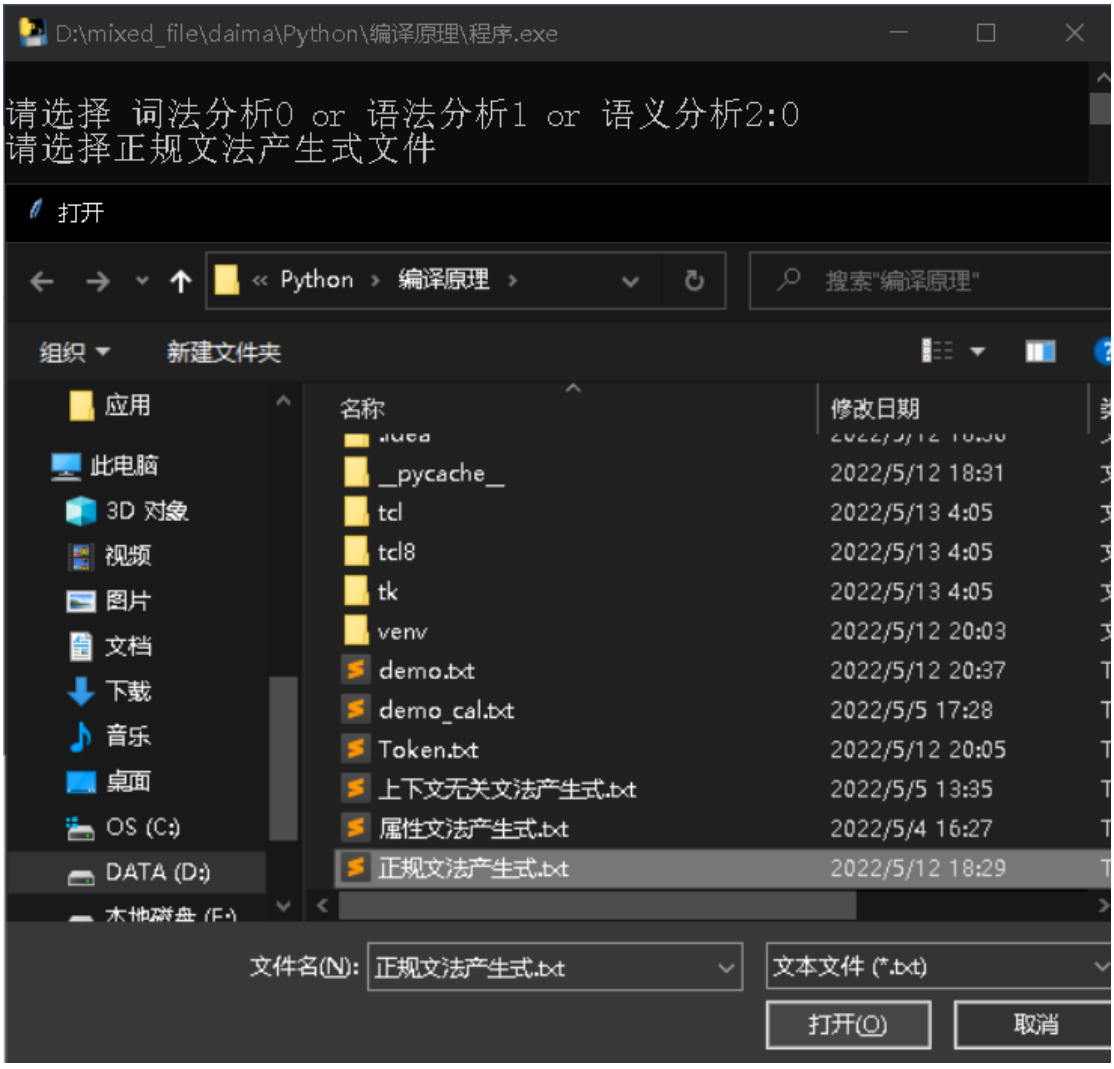
```
5 限定符 ,
5 标识符 m
5 运算符 =
5 常量 3
5 限定符 ;
6 关键词 int
6 标识符 pp
6 限定符 [
6 常量 3
6 限定符 ]
6 运算符 =
6 限定符 {
6 常量 1
6 限定符 ,
6 常量 2
6 限定符 ,
6 常量 3
6 限定符 }
6 限定符 ;
7 关键词 int
7 运算符 *
7 标识符 p
7 运算符 =
7 运算符 &
7 标识符 n
7 限定符 ;
8 关键词 bool
8 标识符 isOK
8 运算符 =
8 关键词 true
8 限定符 ;
9 标识符 n
9 运算符 +=
9 限定符 (
9 常量 3
9 运算符 *
9 标识符 m
9 限定符 ;
10 标识符 myPrint
10 限定符 (
10 标识符 n
10 限定符 ,
10 标识符 m
10 限定符 )
```

```
11 关键词 double
11 标识符 xs
11 运算符 =
11 常量 156.154
11 限定符 ;
12 关键词 double
12 标识符 d
12 运算符 =
12 常量 0.123E+5
12 限定符 ;
13 关键词 double
13 标识符 fs
13 运算符 =
13 限定符 (
13 常量 5+9i
13 限定符 )
13 运算符 +
13 常量 5
13 限定符 ;
14 关键词 string
14 标识符 s
14 运算符 =
14 常量 "hello \"world!"
14 限定符 ;
15 关键词 char
15 标识符 c
15 运算符 =
15 标识符 s
15 限定符 [
15 常量 1
15 限定符 ;
16 关键词 if
16 限定符 (
16 标识符 c
16 运算符 ==
16 常量 'a'
16 限定符 &&
16 限定符 (
16 标识符 n
16 运算符 ==
16 标识符 m
16 限定符 ||
16 标识符 isOK
16 限定符 )
```

```
16 限定符 )
16 限定符 {
17 标识符 n
17 运算符 ++
17 限定符 ;
18 限定符 }
19 关键词 return
19 常量 0
19 限定符 ;
```

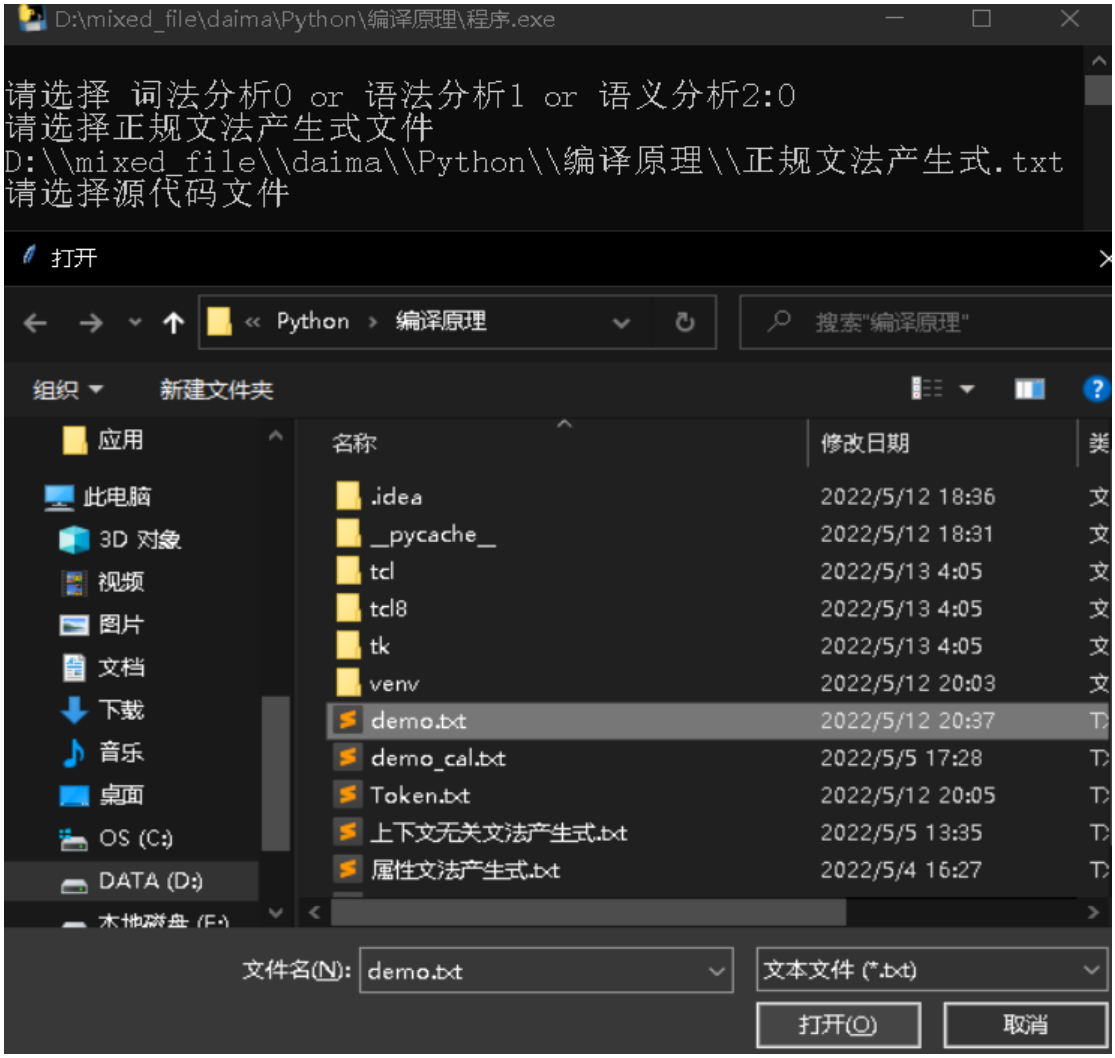
1.4 运行步骤

- 双击可执行文件：程序.exe
- 在命令行窗口根据提示输入:0
- 根据提示在弹出的选择文件框中打开正规文法文件 正规文法产生式.txt，如图 1



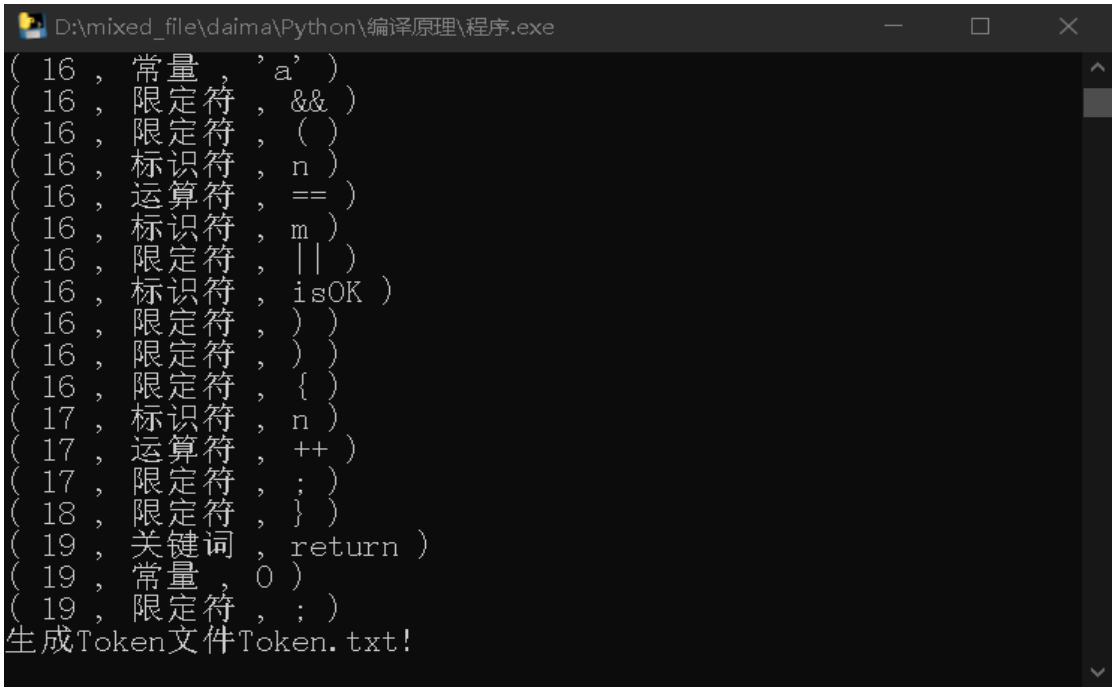
图表 1

- 根据提示在弹出的选择文件框中打开源代码文件 demo.txt，如图 2



图表 2

- 命令行打印出 token 表，如图 3，并生成 Token.txt



图表 3

2 语法分析器

2.1 说明

语法分析器接受一个正规文法文件、一个上下文无关文法文件和一个源代码文件，根据正规文法产生式判断源代码文件中的内容是否符合此文法生成 token 表，再根据上下文无关文法产生式判断 token 表是否符合此文法，若符合，输出 YES，否则输出 NO，并对相关错误进行识别和标记，并打印出相应过程。以及额外生成更加直观的 result.html 文件。

2.2 输入

2.2.1 正规文法文件：正规文法产生式.txt

同 1.2.1

2.2.2 上下文无关文法文件：上下文无关文法产生式.txt

K 表示非终结符集V_N
S 表示开始符 S
sigma 表示终结符集V_T

$\alpha \rightarrow \beta$表示规则 P

$表示$\varepsilon$

```
K:S' S define func_ret func_define paras block para const_add dv_op
sv_op pre_op assigning_op exp logic_exp conjunction define_statement
if_statement else_statement while_statement ret_statement
for_statement func_call ids define_para valuation normal_statement
exps id_add valuation_add para_add exp_add exps
S:S'
sigma:( ) [ ] ; , { } # + - * / & ^ && || | if else return for while
void int double float char string ++ -- ~ # = < > <= >= != == $ id
const
S'->S
S->$
func_ret->void
func_ret->define
```

```
define->int
define->double
define->flout
define->string
define->char
define->bool
define->define *
func_define->func_ret id ( paras ) { block }
func_define->para ( paras ) { block }
func_define->func_ret id ( ) { block }
func_define->para ( ) { block }
para->define id
paras->para para_add
para_add->, paras
para_add->$
assigning_op->=
assigning_op->+=
assigning_op->-=
assigning_op->*=
assigning_op->/=
dv_op->>
dv_op-><
dv_op->==
dv_op-><=
dv_op->>=
dv_op->!=
dv_op->+
dv_op->-
dv_op->*
dv_op->/
dv_op->^
dv_op->&
dv_op->|
sv_op->++
sv_op->--
pre_op->~
define_statement->define valuation valuation_add ;
define_statement->define_para ;
define_para->define ids
valuation->id assigning_op exp
valuation->id [ const ] assigning_op { const const_add }
valuation->id assigning_op { const const_add }
const_add->, const const_add
const_add->$
```

```
valuation_add->, valuation valuation_add
valuation_add->$
exp->exp dv_op exp
exp->exp sv_op
exp->sv_op exp
exp->pre_op exp
exp->func_call
exp->const
exp->( exp )
exp->id
exp->id [ const ]
exp->& id
normal_statement->exp ;
normal_statement->valuation ;
normal_statement->func_call ;
func_call->id ( ids )
func_call->id ( )
ids->id id_add
id_add->, ids
id_add->$
for_statement->for ( define_statement logic_exp ; exp ) { block }
if_statement->if ( logic_exp ) { block } else_statement
if_statement->if ( logic_exp ) { block }
else_statement->else { block }
while_statement->while ( logic_exp ) { block }
ret_statement->return exp ;
conjunction->&&
conjunction->||
logic_exp->exps
exps->exp exp_add
exp_add->conjunction exps
exp_add->$
exp->exp conjunction exp
block->$
block->{ block }
block->define_statement block
block->if_statement block
block->else_statement block
block->for_statement block
block->while_statement block
block->ret_statement block
block->normal_statement block
S->func_define S
S->define_statement S
```

2.2.3 源代码文件：demo.txt

支持简单 C++语法，扩充了复数（形如：1+5i）和科学计数法（形如：0.123E+5）

```
void myPrint(int i,int j){
    return i+j;
}
int main(){
    int n = 5,m = 3;
    int pp[3]={1,2,3};
    int*p=&n;
    bool isOK = true;
    n+=(3*m;
    myPrint(n,m)
    double xs = 156.154;
    double d=0.123E+5;
    double fs=( 5+9i )+5;
    string s = "hello \"world!";
    char c = s[1;
    if(c == 'a'&&(n==m || isOK)){
        n++;
    }
    return 0;
```

2.3 输出

打印出 First 集合，LR(1)预测分析表，LR(1)分析过程，结果 YES or NO，若为 NO，则打印错误所在行列号和错误类型，标记出错误。生成 result.html 文件。用户可进行交互根据下标查询某个项目集。

```
------------------------------------------------First集合------------------------------------------------
FIRST( S' ) = string void int $ double char
FIRST( S ) = $ string int double void char
FIRST( define ) = int string double char
FIRST( func_ret ) = string int void double char
FIRST( func_define ) = string int double void char
FIRST( paras ) = int string double char
FIRST( block ) = id string ~ ++ ( int double { return else while & -- const for $ if char
FIRST( para ) = int string double char
FIRST( const_add ) = , $
FIRST( dv_op ) = > == ^ != < + | * <= - & / >=
FIRST( sv_op ) = ++ --
FIRST( pre_op ) = ~
FIRST( assigning_op ) = =
FIRST( exp ) = id -- ~ & const ( ++
FIRST( logic_exp ) = id -- ~ & const ( ++
FIRST( conjunction ) = || &&
FIRST( define_statement ) = string int double char
FIRST( if_statement ) = if
FIRST( else_statement ) = else
```

图表 4 First 集合

状态	ACTION									
	(	)	[	]	;	,	{	}	#	
0										
1									acc	
2									r_1	
3										
4									r_90	
5										
6									r_91	
7										
8	S_9									
9		S_312								
10		S_11								
11							S_12			
12	S_200						S_16			

图表 5 LR(1)预测分析表

```
----------------------------------------------分析过程----------------------------------------------
步骤:1
状态栈:[0]
符号栈:#
输入栈:['void', 'id', '(', 'int', 'id', ',', 'int', 'id', ')', '{', 'return', 'id', '+', 'id', ';', '}', 'int', 'id', '(', ')', '{', 'int', 'id', '=', 'const',
Action:None
GOTO:341

步骤:2
状态栈:[0, 341]
符号栈:#,void
输入栈:['id', '(', 'int', 'id', ',', 'int', 'id', ')', '{', 'return', 'id', '+', 'id', ';', '}', 'int', 'id', '(', ')', '{', 'int', 'id', '=', 'const', ',', 'i
Action:func_ret->void
GOTO:7

步骤:3
状态栈:[0, 7]
符号栈:#,func_ret
输入栈:['id', '(', 'int', 'id', ',', 'int', 'id', ')', '{', 'return', 'id', '+', 'id', ';', '}', 'int', 'id', '(', ')', '{', 'int', 'id', '=', 'const', ',', 'i
Action:None
GOTO:8
```

图表 6 分析过程

```
NO
0. 9:11: error:except )
    n+=(3*m;
           ^
1. 11:4: error:except ;
    double xs = 156.154;
    ^
2. 15:16: error:except ]
    char c = s[1;
                ^
3. 19:12: error:except }
    return 0;            ^
生成结果文件result.html!
```

图表 7 结果

```
请输入想获得的项目集：8

----------------------------------------------状态8----------------------------------------------
func_define->func_ret id · ( paras ) { block } , int/void/string/#/char/double/$
func_define->func_ret id · ( ) { block } , int/void/string/#/char/double/$

----------------------------------------------子结点----------------------------------------------
->          (           -> 9

----------------------------------------------结束----------------------------------------------
```

图表 8 查询项目集

源代码

```
void myPrint(int i,int j){
    return i+j;
}
int main(){
    int n = 5,m = 3;
    int*pp={1,2,3};
    int*p=&n;
    bool isOK = true;
    n+=(3*m;
    myPrint(n,m)
    double xs = 156.154;
    double d=0.123E+5;
    double fs=( 5+9i )+5;
    string s = "hello \"world!";
    char c = s[1;
```

Token表

```
1 关键词 void
1 标识符 myPrint
1 限定符 (
1 关键词 int
1 标识符 i
1 限定符 ,
1 关键词 int
1 标识符 j
1 限定符 )
1 限定符 {
2 关键词 return
2 标识符 i
2 运算符 +
2 标识符 j
2 限定符 ;
```

FIRST集合

```
FIRST( S' )=double  void  string  $  int  char
FIRST( S )=void  double  string  $  int  char
FIRST( define )=double  string  int  char
FIRST( func_ret )=double  void  string  int  char
FIRST( func_define )=void  double  string  int  char
FIRST( paras )=double  string  int  char
FIRST( block )=for  --  char  {  id  return  ~  double  const  &  if  string  (  $  ++  while  else  int
FIRST( para )=double  string  int  char
```

LR1预测分析表

状态	(	)	[	]	;	,	{	}	#	+	-	*	/	&	^	&&	||	|	if	else	return
0																					
1									acc												
2									r_1												
3																					
4									r_90												
5																					
6									r_91												
7																					
8	S_9																				
9		S_306																			
10		S_11																			

LR1预测分析过程

```
步骤  1
状态栈 [0]
符号栈 #
输入栈 ['void', 'id', '(', 'int', 'id', ',', 'int', 'id', ')', '{', 'return', 'id', '+', 'id', ';', '}', 'int', 'id', '(', ')', '{', 'int', 'id', '=', 'const', ',', 'id', '=', 'const', ';', 'int', '*', 'id', '=', '{', 'const', ',', 'const', ',', 'const', '}', ';', 'int', '*', 'id', '=', '&', 'id', ';', '+=', '(', 'const', '*', 'id', ';', 'id', '(', 'id', ',', 'id', ')', 'double', 'id', '=', 'const', ';', 'double', 'id', '=', '(', 'const', ')', '+', 'const', ',', 'string', 'id', '=', 'const', ';', 'char', 'id', '=', 'id', '[', 'const', ';', 'i
'(', 'id', '==', 'id', ']|', 'id', ')', ';', '{', 'id', '++', ';', '}', 'return', 'const', ';', '#']
Action None
GOTO 335
```

语法分析结果

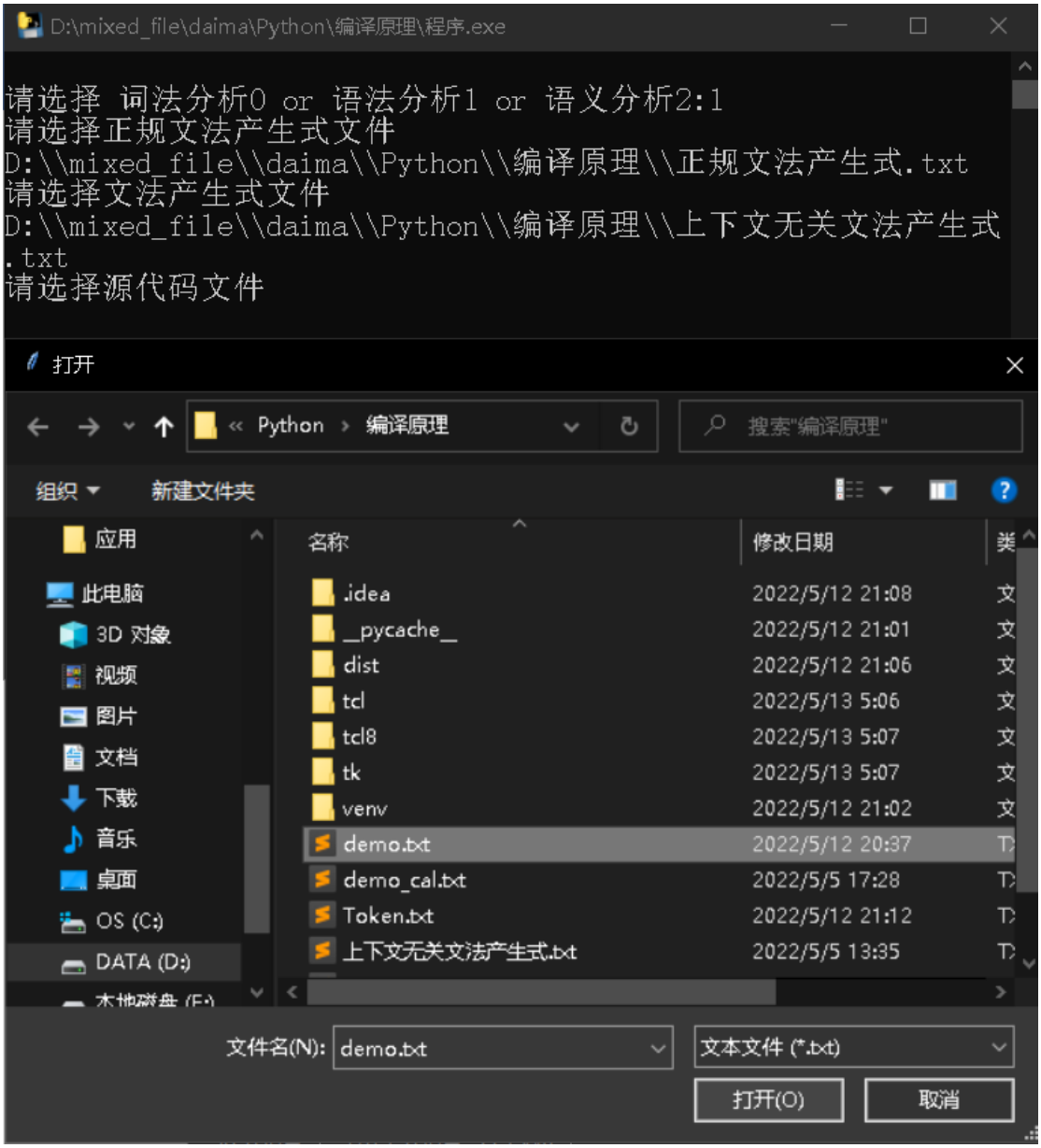
NO

```
9:11: error:except )
    n+=(3*m;
          ^
11:4: error:except ;
    double xs = 156.154;
    ^
15:16: error:except ]
    char c = s[1;
```

图表 9result.html

2.4 运行步骤

- 双击可执行文件：程序.exe
- 在命令行窗口根据提示输入:1
- 根据提示在弹出的选择文件框中打开正规文法文件 正规文法产生式.txt
- 根据提示在弹出的选择文件框中打开上下文无关文法文件 上下文无关文法产生式.txt
- 根据提示在弹出的选择文件框中打开源代码文件 demo.txt，如图 10



图表 10

- 命令行输出 First 集合，LR(1)预测分析表，LR(1)分析过程，结果 YES or NO，若为 NO，则输出错误所在行列号和错误类型，标记出错误。如图 11
- 生成更加直观的 result.html 文件
- 用户可以输入项目集下标对某个项目集进行查询。输入-1 即可退出。

图表 11

3 语义分析器

3.1 说明

语法分析器接受一个正规文法文件、一个属性文法文件和一个表达式文件，根据正规文法产生式判断表达式文件中的内容是否符合此文法生成 token 表，再根据属性文法产生式判断 token 表是否符合此文法，若符合，输出计算结果和四元式，否则输出 NO，并对相关错误进行识别和标记，打印出相应过程。以及额外生成更加直观的 result.html 文件。

3.2 输入

3.2.1 正规文法文件：正规文法产生式.txt

同 1.2.1

3.2.2 属性文法文件：属性文法产生式.txt

K 表示非终结符集V_N
S 表示开始符 S
sigma 表示终结符集V_T

$\alpha \to \beta$表示规则 P　{}表示语义动作

```
K:S E T F R Q
S:S
sigma:+ * ( ) const - /
S->E{print(E.val)}
E->E + T{E.val:=E.val+T.val}
E->T{E.val:=T.val}
T->T * Q{T.val:=T.val*Q.val}
T->Q{T.val:=Q.val}
Q->Q / R{Q.val:=Q.val/R.val}
Q->R{Q.val:=R.val}
R->R - F{R.val:=R.val-F.val}
R->F{R.val:=F.val}
F->( E ){F.val:=E.val}
F->const{F.val:=const.lexval}
```

3.2.3 表达式文件：expression.txt

支持加减乘除运算

```
5+2*(4-1)+8/(4-2)
```

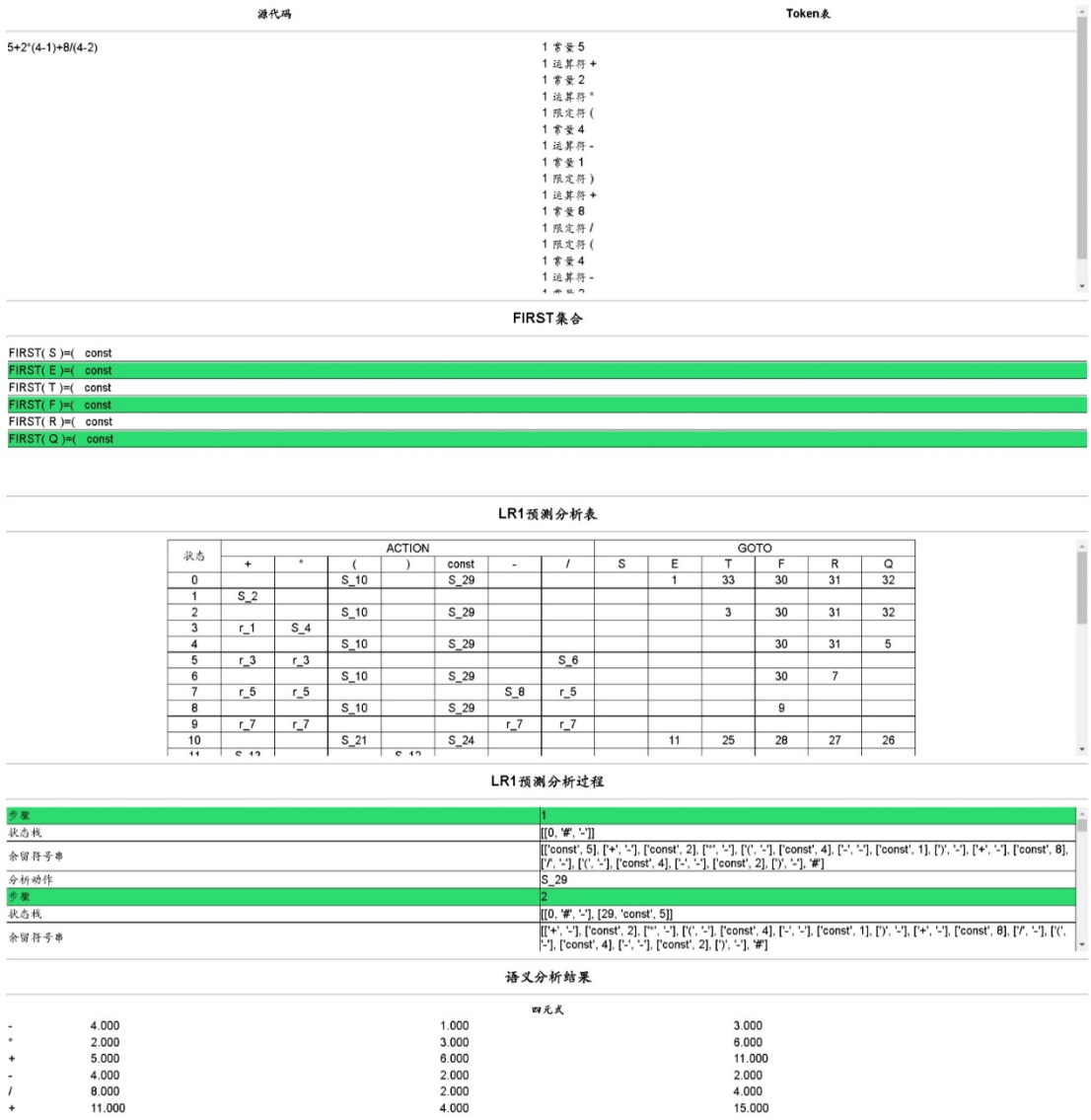
3.3 输出

打印出 First 集合，LR(1)预测分析表，LR(1)分析过程，结果计算值和四元组 or NO，若为 NO，则打印错误所在行列号和错误类型，标记出错误。生成 result.html 文件。用户可进行交互根据下标查询某个项目集。
四元组形式表示为(op, y, z,x)，其中 op 为运算符，y 和 z 为运算数，x 为运算结果。
First 集合，LR(1)预测分析表，LR(1)分析过程，同 2.3

```
15.0

-----------------------------------------------四元式-----------------------------------------------
(- , 4.000 , 1.000 , 3.000)
(* , 2.000 , 3.000 , 6.000)
(+ , 5.000 , 6.000 , 11.000)
(- , 4.000 , 2.000 , 2.000)
(/ , 8.000 , 2.000 , 4.000)
(+ , 11.000 , 4.000 , 15.000)
生成结果文件result.html!
```

图表 12 结果



源代码

5+2*(4-1)+8/(4-2)

Token表

1 常量 5
1 运算符 +
1 常量 2
1 运算符 *
1 限定符 (
1 常量 4
1 运算符 -
1 常量 1
1 限定符)
1 运算符 +
1 常量 8
1 限定符 /
1 限定符 (
1 常量 4
1 运算符 -

FIRST集合

FIRST(S)=(const
FIRST(E)=(const
FIRST(T)=(const
FIRST(F)=(const
FIRST(R)=(const
FIRST(Q)=(const

LR1预测分析表

状态	ACTION							GOTO					
	+	*	(	)	const	-	/	S	E	T	F	R	Q
0			S_10		S_29				1	33	30	31	32
1	S_2												
2			S_10		S_29					3	30	31	32
3	r_1	S_4											
4			S_10		S_29						30	31	5
5	r_3	r_3					S_6						
6			S_10		S_29						30	7	
7	r_5	r_5				S_8	r_5						
8			S_10		S_29						9		
9	r_7	r_7				r_7	r_7						
10			S_21		S_24				11	25	28	27	26

LR1预测分析过程

步骤	1
状态栈	[[0, '#', '-']]
余留符号串	[['const', 5], ['+', '-'], ['const', 2], ['*', '-'], ['(', '-'], ['const', 4], ['-', '-'], ['const', 1], [')', '-'], ['+', '-'], ['const', 8], ['/', '-'], ['(', '-'], ['const', 4], ['-', '-'], ['const', 2], [')', '-'], '#']
分析动作	S_29
步骤	2
状态栈	[[0, '#', '-'], [29, 'const', 5]]
余留符号串	[['+', '-'], ['const', 2], ['*', '-'], ['(', '-'], ['const', 4], ['-', '-'], ['const', 1], [')', '-'], ['+', '-'], ['const', 8], ['/', '-'], ['(', '-'], ['const', 4], ['-', '-'], ['const', 2], [')', '-'], '#']

语义分析结果

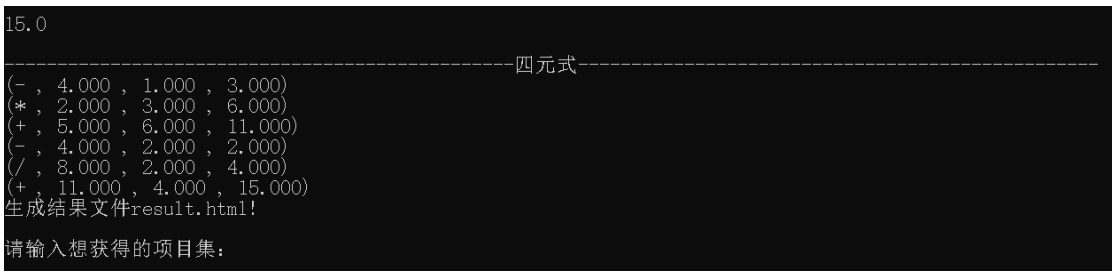
四元式

-	4.000	1.000	3.000
*	2.000	3.000	6.000
+	5.000	6.000	11.000
-	4.000	2.000	2.000
/	8.000	2.000	4.000
+	11.000	4.000	15.000

图表 13result.html

3.4 运行步骤

- 双击可执行文件：程序.exe
- 在命令行窗口根据提示输入:2
- 根据提示在弹出的选择文件框中打开正规文法文件 正规文法产生式.txt
- 根据提示在弹出的选择文件框中打开属性文法文件 属性文法产生式.txt
- 根据提示在弹出的选择文件框中打开源代码文件 expression.txt
- 命令行输出 First 集合，LR(1)预测分析表，LR(1)分析过程，结果计算值和四元组 or NO，若为 NO，则打印错误所在行列号和错误类型，标记出错误。如图 14
- 生成更加直观的 result.html 文件
- 用户可以输入项目集下标对某个项目集进行查询。输入-1 即可退出。


15.0
----------四元式----------
(- , 4.000 , 1.000 , 3.000)
(* , 2.000 , 3.000 , 6.000)
(+ , 5.000 , 6.000 , 11.000)
(- , 4.000 , 2.000 , 2.000)
(/ , 8.000 , 2.000 , 4.000)
(+ , 11.000 , 4.000 , 15.000)
生成结果文件result.html!
请输入想获得的项目集：

图表 14