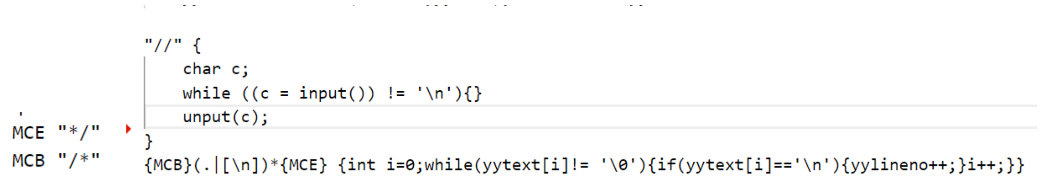
Compiler project phase 1

吕美欧 12111645
张艺严 12011528

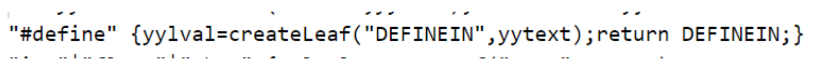
1. 单行注释和多行注释



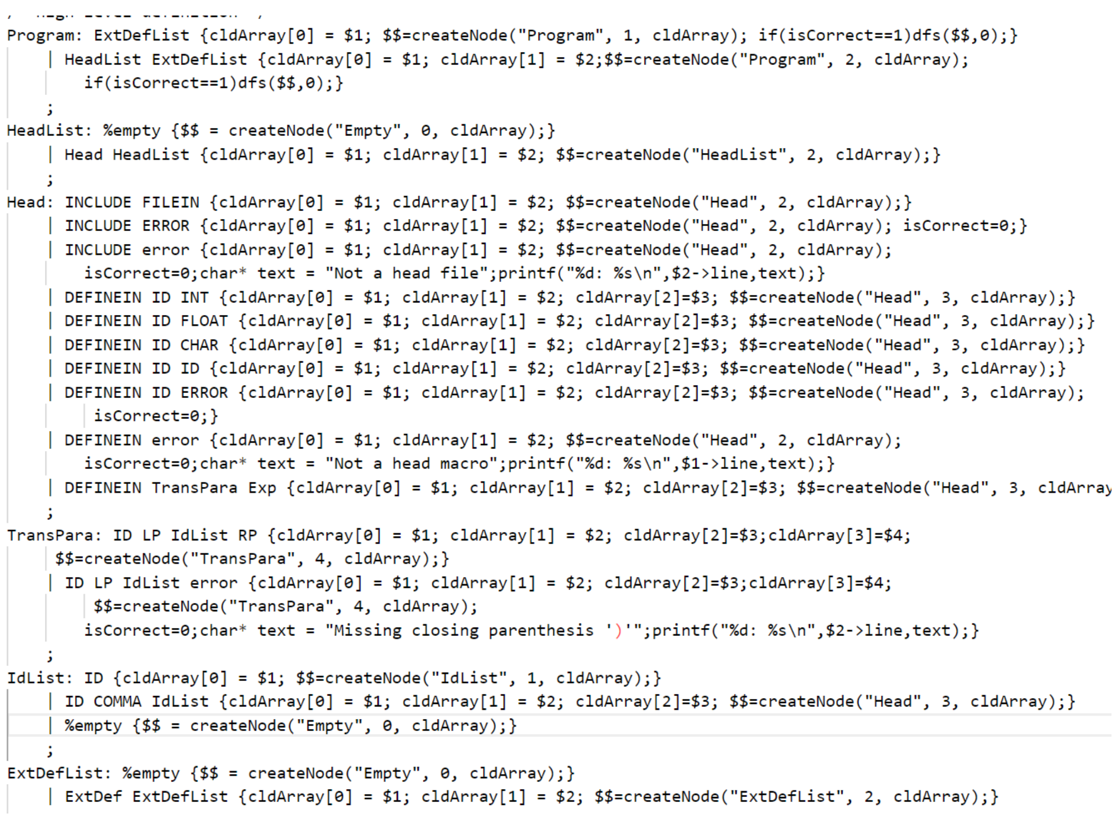
```
                "//" {
                    char c;
                    while ((c = input()) != '\n'){}
                    unput(c);
MCE "*/"        }
MCB "/*"        {MCB}(.|[\n])*{MCE} {int i=0;while(yytext[i]!= '\0'){if(yytext[i]=='\n'){yylineno++;}i++;}}
```

单行注释：当遇到”//”后，用 char c 从之后的缓冲区中获取每一个字符，如果不是换行，就一直消耗字符，当是换行符时结束循环，并把这个换行符重新放回缓冲区。多行注释：字符两头是/* */ 中间是任意长字符或换行符时，遍历这串字符，遇到字符为换行符时记录行数的 yylineo 增加。

2. 宏预处理器



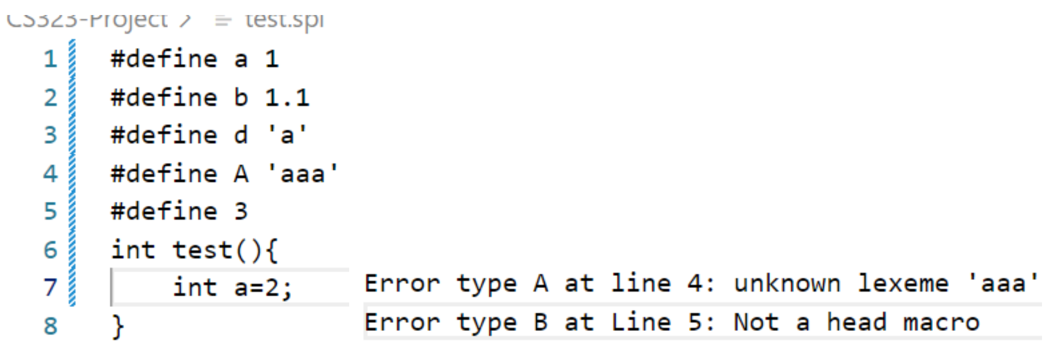
```
"#define" {yylval=createLeaf("DEFINEIN",yytext);return DEFINEIN;}
```



```
Program: ExtDefList {cldArray[0] = $1; $$=createNode("Program", 1, cldArray); if(isCorrect==1)dfs($$,0);}
    | HeadList ExtDefList {cldArray[0] = $1; cldArray[1] = $2;$$=createNode("Program", 2, cldArray);
      if(isCorrect==1)dfs($$,0);}
    ;
HeadList: %empty {$$ = createNode("Empty", 0, cldArray);}
    | Head HeadList {cldArray[0] = $1; cldArray[1] = $2; $$=createNode("HeadList", 2, cldArray);}
    ;
Head: INCLUDE FILEIN {cldArray[0] = $1; cldArray[1] = $2; $$=createNode("Head", 2, cldArray);}
    | INCLUDE ERROR {cldArray[0] = $1; cldArray[1] = $2; $$=createNode("Head", 2, cldArray); isCorrect=0;}
    | INCLUDE error {cldArray[0] = $1; cldArray[1] = $2; $$=createNode("Head", 2, cldArray);
      isCorrect=0;char* text = "Not a head file";printf("%d: %s\n",$2->line,text);}
    | DEFINEIN ID INT {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3; $$=createNode("Head", 3, cldArray);}
    | DEFINEIN ID FLOAT {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3; $$=createNode("Head", 3, cldArray);}
    | DEFINEIN ID CHAR {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3; $$=createNode("Head", 3, cldArray);}
    | DEFINEIN ID ID {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3; $$=createNode("Head", 3, cldArray);}
    | DEFINEIN ID ERROR {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3; $$=createNode("Head", 3, cldArray);
       isCorrect=0;}
    | DEFINEIN error {cldArray[0] = $1; cldArray[1] = $2; $$=createNode("Head", 2, cldArray);
      isCorrect=0;char* text = "Not a head macro";printf("%d: %s\n",$1->line,text);}
    | DEFINEIN TransPara Exp {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3; $$=createNode("Head", 3, cldArray
    ;
TransPara: ID LP IdList RP {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3;cldArray[3]=$4;
     $$=createNode("TransPara", 4, cldArray);}
    | ID LP IdList error {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3;cldArray[3]=$4;
       $$=createNode("TransPara", 4, cldArray);
      isCorrect=0;char* text = "Missing closing parenthesis ')'";printf("%d: %s\n",$2->line,text);}
    ;
IdList: ID {cldArray[0] = $1; $$=createNode("IdList", 1, cldArray);}
    | ID COMMA IdList {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3; $$=createNode("Head", 3, cldArray);}
    | %empty {$$ = createNode("Empty", 0, cldArray);}
    ;
ExtDefList: %empty {$$ = createNode("Empty", 0, cldArray);}
    | ExtDef ExtDefList {cldArray[0] = $1; cldArray[1] = $2; $$=createNode("ExtDefList", 2, cldArray);}
```

整个程序分为两部分，一部分是头区，另一部分是代码区。头区的宏有两种形式，一种是 #define PI 3.14 另一种是 #define S(a,b) a+b。

对于第一种，结构需为 #define ID Float/Int/Char/ID ，如果#define 后面不是 ID，会报 Not a head macro，如果 float/int/char 有误会报 A 类错误，如：

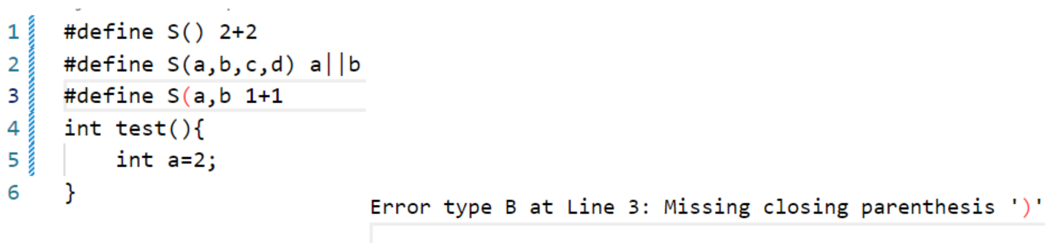


```
CS323-Project > test.spl
#define a 1
#define b 1.1
#define d 'a'
#define A 'aaa'
#define 3
int test(){
    int a=2;
}
```

```
Error type A at line 4: unknown lexeme 'aaa'
Error type B at Line 5: Not a head macro
```

另一种结构 #define ID (ID , ID) Expression，(ID , ID) 这部分在代码中的标签是 TransPara，它支持任意长度的传参，exp 和后面一致，可以是运算，也可以是逻辑。同时它也支持丢括号的错误检测。

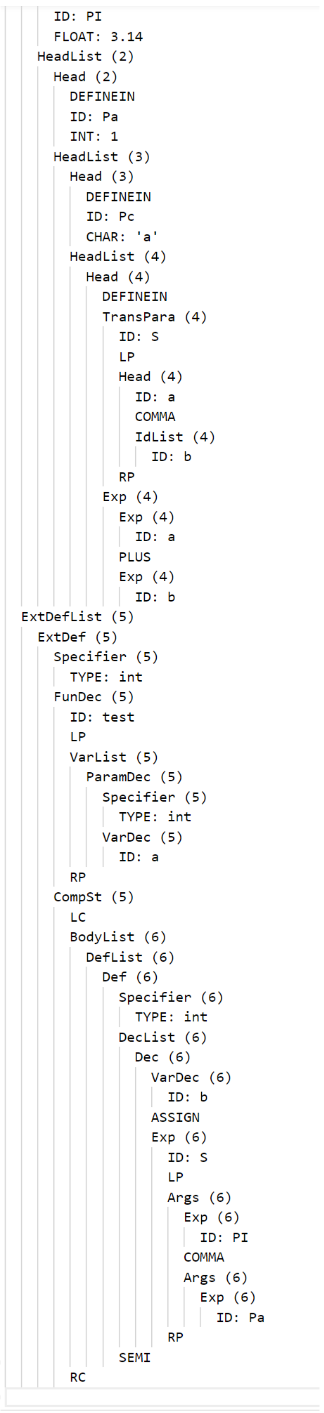
如：



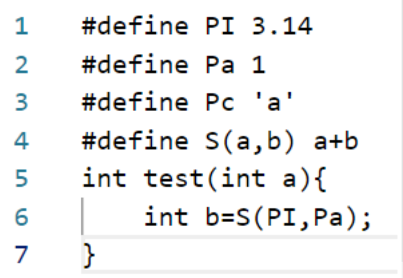
```
#define S() 2+2
#define S(a,b,c,d) a||b
#define S(a,b 1+1
int test(){
    int a=2;
}
```

```
Error type B at Line 3: Missing closing parenthesis ')'
```

正确时树结构如下：

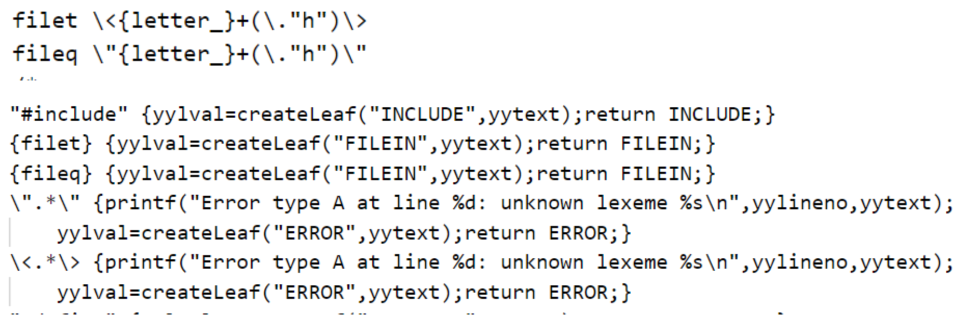


```
    ID: PI
    FLOAT: 3.14
  HeadList (2)
    Head (2)
      DEFINEIN
      ID: Pa
      INT: 1
    HeadList (3)
      Head (3)
        DEFINEIN
        ID: Pc
        CHAR: 'a'
      HeadList (4)
        Head (4)
          DEFINEIN
          TransPara (4)
            ID: S
            LP
            Head (4)
              ID: a
              COMMA
              IdList (4)
                ID: b
            RP
          Exp (4)
            Exp (4)
              ID: a
            PLUS
            Exp (4)
              ID: b
ExtDefList (5)
  ExtDef (5)
    Specifier (5)
      TYPE: int
    FunDec (5)
      ID: test
      LP
      VarList (5)
        ParamDec (5)
          Specifier (5)
            TYPE: int
          VarDec (5)
            ID: a
      RP
    CompSt (5)
      LC
      BodyList (6)
        DefList (6)
          Def (6)
            Specifier (6)
              TYPE: int
            DecList (6)
              Dec (6)
                VarDec (6)
                  ID: b
                ASSIGN
                Exp (6)
                  ID: S
                  LP
                  Args (6)
                    Exp (6)
                      ID: PI
                    COMMA
                    Args (6)
                      Exp (6)
                        ID: Pa
                  RP
            SEMI
      RC
```



```
#define PI 3.14
#define Pa 1
#define Pc 'a'
#define S(a,b) a+b
int test(int a){
    int b=S(PI,Pa);
}
```

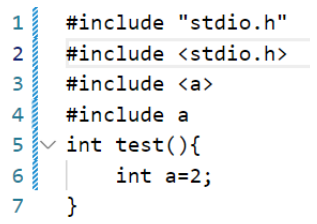
3. 文件包含



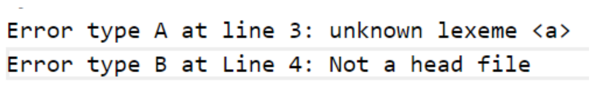
```
filet \<{letter_}+(\."h")\>
fileq \"{letter_}+(\."h")\"

"#include" {yylval=createLeaf("INCLUDE",yytext);return INCLUDE;}
{filet} {yylval=createLeaf("FILEIN",yytext);return FILEIN;}
{fileq} {yylval=createLeaf("FILEIN",yytext);return FILEIN;}
\".*\" {printf("Error type A at line %d: unknown lexeme %s\n",yylineno,yytext);
    yylval=createLeaf("ERROR",yytext);return ERROR;}
\<.*\> {printf("Error type A at line %d: unknown lexeme %s\n",yylineno,yytext);
    yylval=createLeaf("ERROR",yytext);return ERROR;}
```

Filet 是<stdio.h>类型，fileq 是"stdio.h"类型，不符合这种格式的会被认为是错误。

Bison 文件和前面的宏在一个区域，因此不再粘贴代码。如果不是<x.x>或"x.x"格式会被认为是 A 类错误，如果引入文件的地方错误填写数字或普通 ID 会被认为是 B 类错误。

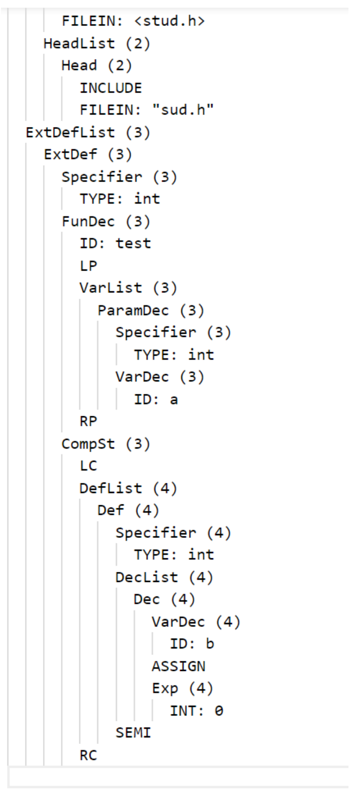


```
#include "stdio.h"
#include <stdio.h>
#include <a>
#include a
int test(){
    int a=2;
}
```

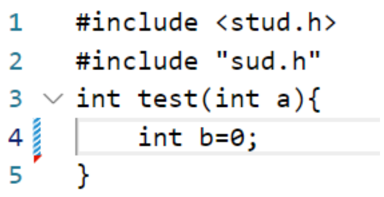


```
Error type A at line 3: unknown lexeme <a>
Error type B at Line 4: Not a head file
```

以下是正确运行时的树：

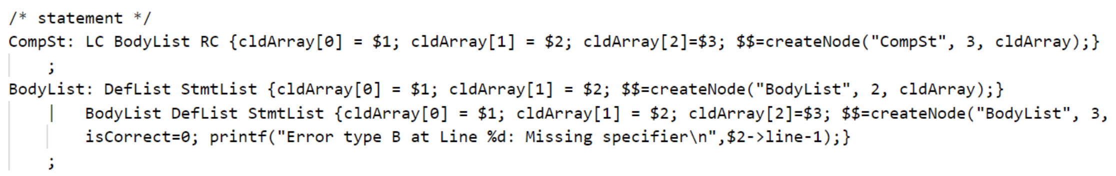


```
    FILEIN: <stud.h>
  HeadList (2)
    Head (2)
      INCLUDE
      FILEIN: "sud.h"
ExtDefList (3)
  ExtDef (3)
    Specifier (3)
      TYPE: int
    FunDec (3)
      ID: test
      LP
      VarList (3)
        ParamDec (3)
          Specifier (3)
            TYPE: int
          VarDec (3)
            ID: a
      RP
    CompSt (3)
      LC
      DefList (4)
        Def (4)
          Specifier (4)
            TYPE: int
          DecList (4)
            Dec (4)
              VarDec (4)
                ID: b
              ASSIGN
              Exp (4)
                INT: 0
          SEMI
      RC
```



```
#include <stud.h>
#include "sud.h"
int test(int a){
    int b=0;
}
```

4：其他说明



```
/* statement */
CompSt: LC BodyList RC {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3; $$=createNode("CompSt", 3, cldArray);}
    ;
BodyList: DefList StmtList {cldArray[0] = $1; cldArray[1] = $2; $$=createNode("BodyList", 2, cldArray);}
    |   BodyList DefList StmtList {cldArray[0] = $1; cldArray[1] = $2; cldArray[2]=$3; $$=createNode("BodyList", 3,
        isCorrect=0; printf("Error type B at Line %d: Missing specifier\n",$2->line-1);}
    ;
```

为了能识别 statement 在 definition 前边的情况，在这里多加了一层。

使用方法：./splc ../test-ex(目录名)/test_2.spl(文件名) 结果会直接打印在终端