

12011619 Liquan Wang
 12111744 Wenhui Tao
 12111611 Ruixiang Jiang

1 cs323-project-phase1

catalogue

1 cs323-project-phase1	1
1.1 General introduction	1
1.1.1 How To Run It	1
1.1.2 FLEX part	1
1.1.3 BISON part	2
1.2 Bonus	2
1.2.1 single- and multi-line comment	2
1.2.2 macro preprocessor and file inclusion	3
1.2.3 for statements	6

1.1 General introduction

Completion status:

Pass all basic sample tests, and add following features:

1.1.1 How To Run It

1. clean the environment with the make clean command
2. Use make in the root directory
3. make test Indicates a test example

1.1.2 FLEX part

- yylval returns the name \n of the lexical unit
- We detect tokens in the correct format and tokens in the wrong format. The correct token can help the bison part to do syntax analysis, and the wrong token can also help the bison to do error recovery and analyze the entire program.
- Use `%macro;` state to distinguish macro recognition

1.1.3 BISON part

- The various operators are defined from top to bottom in order of precedence from low to high. The priority is specified by text % prec to eliminate some ambiguity.
- For each production expression, we use the custom fun “concat_shift” to truncate the next level of string by text\n and then add two Spaces before each part. When recursively to Program, output parsing tree.

1.2 Bonus

Please note:

Because of aesthetic issues, the code is partially commented, but the actual program is not commented

To make the code more intuitive, we’ve attached test samples and results that you can skip if you don’t want to read them

- single- and multi-line comment
- macro preprocessor
- file inclusion
- for statements

In addition, there is more complete error detection, which is reflected in our own basic test sample (not in test-ex).

1.2.1 single- and multi-line comment

Listing 1: Implementation

```
1  /* in lex.l
2  Recognize but do not react to ignore comments*/
3  "//" .* {}
4  "/*" ((( "/*" [^/] ) ? ) | [^*] ) * "*/" {}
```

Listing 2: test

```
1  // this is a single Line
2  /* here is a A multi-line comment */
3  /* here is a /* A illegal*/ multi-line comment */
```

Listing 3: result

```
1 Program (1)
```

1.2.2 macro preprocessor and file inclusion

Listing 4: lex.l Implementation

```
1  /* in lex.l
2  Enter the <macro> state after a special beginning
   is recognized to distinguish it from the
   exception recognition capture
3  */
4  "#include" {yylval=strdup("INCLUDE\n"); BEGIN(
   macro); return INCLUDE;}
5  "#define" {yylval=strdup("DEFINE\n"); BEGIN(macro)
   ; return DEFINE;}
6  "#ifdef" {yylval=strdup("IFDEF\n"); BEGIN(macro);
   return IFDEF;}
7  "#else" {yylval=strdup("MACROELSE\n"); return
   MACROELSE;}
8  "#endif" {yylval=strdup("ENDIF\n"); return ENDIF;}
9  <macro>{
10     "," {yylval=strdup("COMMA\n");return COMMA;}
11     "<" {return LT;}
12     ">" {return GT;}
13     "\" {return DQUOTE;}
14     "\n" {BEGIN(INITIAL);}
15     [ \t]+ /*ignore word splits*/{}
16     // [^," "<>"\n]+ {asprintf(&yylval,"%s\n",yytext);
   return MACRO;}
17 }
```

Listing 5: syntax.y Implementation

```
1  /* in syntax.y
2  Macro, file introduction automatically changes to
   the beginning of the file, and parallel output,
   to achieve #define #ifdef #else #endif #
   include
3  */
```

```
4  /*You can define multiple macros on a single line
   separated by commas*/
5
6  RES:
7  Program
8  | MACROStmt Program
9
10 /* #define && #include*/
11 MACROStmt:
12     INCLUDEStmt
13 | DEFINEStmt
14 | DEFINEStmt MACROStmt
15 | INCLUDEStmt MACROStmt
16
17 INCLUDEStmt:
18     INCLUDE LT MACRO GT
19 | INCLUDE DQUOT MACRO DQUOT
20
21 DEFINEStmt:
22     DEFINE MACRO MACRO
23 | DEFINEStmt COMMA MACRO MACRO
24
25 /*#ifdef #else #endif*/
26 Stmt:
27     ....
28 | IFDEF Stmt ENDIF
29 | IFDEF MACRO Stmt MACROELSE Stmt ENDIF
```

Listing 6: test

```
1  #define MACRO_NAME replacement_text , MACRO_NAME2
   replacement_text2
2  #define MACRO_NAME1 replacement_text1
3  #include <header_file.h>
4  #include "header_file.h"
5  #define DEBUG 1
6
7  int main() {
8  #ifdef DEBUG
9      printf('y');
```

```
10 #else
11     printf('n');
12 #endif
13
14     return 0;
15 }
```

Listing 7: result

```
1  DEFINE (1)
2      MACRO_NAME
3      replacement_text
4  DEFINE (1)
5      MACRO_NAME2
6      replacement_text2
7  DEFINE (2)
8      MACRO_NAME1
9      replacement_text1
10 INCLUDE (3)
11     header_file.h
12 INCLUDE (4)
13     header_file.h
14 DEFINE (5)
15     DEBUG
16     1
17 Program (7)
18     ...
19         Stmt (8)
20             IFDEF
21             DEBUG
22             Stmt (9)
23                 Exp (9)
24                     ID: printf
25                     LP
26                     Args (9)
27                         Exp (9)
28                             CHAR: 'y'
29                     RP
30             SEMI
31         MACROELSE
```

```

32          Stmt (11)
33          Exp (11)
34          ID: printf
35          LP
36          Args (11)
37          Exp (11)
38          CHAR: 'n'
39          RP
40          SEMI
41          ...

```

1.2.3 for statements

Listing 8: lex.l Implementation

```

1  for {yylval=strdup("FOR\n"); return FOR;}

```

Listing 9: syntax.y Implementation

```

1  /*The logic of for is very similar to "if" "while"*/
2  Stmt:
3  ...
4  | FOR LP Exp SEMI Exp SEMI Exp RP Stmt {asprintf(&
    $$,"Stmt_␣(%d)\n%s\n", @$.first_line,
    concat_shift($1,$2,$3,$4,$5,$6,$7,$8,$9));}

```

Listing 10: FOR test

```

1  int test_2(int a, int b)
2  {
3      for (i = 0; i < 5; i=i+1) {
4          printf('now␣it␣is␣d␣\n', i);
5      }
6      return 0;
7  }

```

Listing 11: FOR test result

```

1  Program (1)
2      ...
3      FOR

```

4	LP
5	Exp (3)
6	Exp (3)
7	ID: i
8	ASSIGN
9	Exp (3)
10	INT: 0
11	SEMI
12	Exp (3)
13	Exp (3)
14	ID: i
15	LT
16	Exp (3)
17	INT: 5
18	SEMI
19	Exp (3)
20	Exp (3)
21	ID: i
22	ASSIGN
23	Exp (3)
24	Exp (3)
25	ID: i
26	PLUS
27	Exp (3)
28	INT: 1
29	RP
30	Stmt (3)
31	...