12011619 Liquan Wang
12111744 Wenhui Tao
12111611 Ruixiang Jiang

1 cs323-project-phase1

catalogue

1.1 General introduction

Completion status：

Pass all basic sample tests，and finish some additionally supported 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

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

Listing 3: result

```
  Program (1)
```

1.2.2 macro preprocessor and file inclusion

Listing 4: lex.l Implementation

```
  /* in lex.l
  Enter the <macro> state after a special beginning
     is recognized to distinguish it from the
     exception recognition capture
  */
  "#include" {yylval=strdup("INCLUDE\n"); BEGIN(
     macro); return INCLUDE;}
  "#define" {yylval=strdup("DEFINE\n"); BEGIN(macro)
     ; return DEFINE;}
  "#ifdef" {yylval=strdup("IFDEF\n"); BEGIN(macro);
     return IFDEF;}
  "#else" {yylval=strdup("MACROELSE\n"); return
     MACROELSE;}
  "#endif" {yylval=strdup("ENDIF\n"); return ENDIF;}
  <macro>{
    "," {yylval=strdup("COMMA\n");return COMMA;}
    "<" {return LT;}
    ">" {return GT;}
//  \"  {return DQUOT;}
    \n  {BEGIN(INITIAL);}
    [ \t]+ /*ignore word splits*/{}
//  [^," "<>"\n]+ {asprintf(&yylval,"%s\n",yytext);
   return MACRO;}
}
```

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>
```

```c
#include "header_file.h"
#define DEBUG 1

int main() {
#ifdef DEBUG
    printf('y');
#else
    printf('n');
#endif

    return 0;
}
```

Listing 7: result

```
DEFINE (1)
  MACRO_NAME
  replacement_text
DEFINE (1)
  MACRO_NAME2
  replacement_text2
DEFINE (2)
  MACRO_NAME1
  replacement_text1
INCLUDE (3)
  header_file.h
INCLUDE (4)
  header_file.h
DEFINE (5)
  DEBUG
  1
Program (7)
  ...
          Stmt (8)
            IFDEF
            DEBUG
            Stmt (9)
              Exp (9)
                ID: printf
                LP
```

```
                  Args (9)
                    Exp (9)
                      CHAR: 'y'
                  RP
                SEMI
              MACROELSE
              Stmt (11)
                Exp (11)
                  ID: printf
                  LP
                  Args (11)
                    Exp (11)
                      CHAR: 'n'
                  RP
                SEMI
    ...
```

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('i', i);
5      }
6      return 0;
7  }
```

Listing 11: FOR test result

```
Program (1)
    ...
    FOR
    LP
    Exp (3)
      Exp (3)
        ID: i
      ASSIGN
      Exp (3)
        INT: 0
    SEMI
    Exp (3)
      Exp (3)
        ID: i
      LT
      Exp (3)
        INT: 5
    SEMI
    Exp (3)
      Exp (3)
        ID: i
      ASSIGN
      Exp (3)
        Exp (3)
          ID: i
        PLUS
        Exp (3)
          INT: 1
    RP
    Stmt (3)
    ...
```