CS323 Complier Phase1 Report

Partner and task division:

贺晗 12012013: flex

郑鑫颖 11912039: construct and print bison parsing tree & error recovery

张海涵 12012222: bonus & error recovery

Part 1 flex

We focus on the implementation as follows:

1. we carefully order each pattern so that each lexeme can be correctly identified.
2. **Float**: `{digit}{digit}*"\."{digit}{digit}*`
3. **ID**: `{letter_}({letter_}|{digit})*`. A similar regular expression `{digit}({letter_}|{digit})*` is added to identify these invalid IDs which begins with digits.
4. **INT**: `{digit}*|("0x"|"0X"){hex}{hex}*` . Then we handled the situation that has unnecessary 0s in the beginning in c codes.
5. **Char**: `\'.*\'` is used to identify all lexemes between a single quotation. Then check whether the lexeme is valid, that is it should be a single character or in the form of `0x{hex}{hex}`
6. For the wrong format lexemes, we return a **Fault** token.

Part 2 bison

Our implementation is as follows:

1. **Constructing and printing the parse tree.**
    1. we construct **tree nodes** for terminal and non-terminal tokens.

```
VarDec: VarDec LB INT RB
{
$$ = insert("VarDec",4,$1,alloNodeC("
[","LB"),alloNodeI($3,"INT"),alloNodeC("]","RB"));
@$ = @1;$$->lineNo=(@1).first_line;
}
```

```
2.  When the construction finishes, we print the tree.
```

```
Program : Headers ExtDefList {
    $$ = insert("Program",2,$1,$2);
    @$ = @1;$$->lineNo=(@1).first_line;
    if(flag ==0){printTree($$,0);}};
```

2. **Error recovery.**
    1. we insert some error tokens in the right place to do error recovery.

2. we define a global variable "flag", and set it to 1 if errors are detected. Only when the flag equals 0, then we print the tree
3. we detect the following errors:

```
VarDec: FAULT error => unknown lexeme;
      | VarDec LB error RB => Error type B: Wrong type of index;
FunDec: ID LP error => Error type B: Missing closing parenthesis;
CompSt: LC DefList StmtList error => Error type B: Missing specifier;
Stmt: RETURN Exp error => Error type B: Missing semicolon;
    | Exp error => Error type B: Missing semicolon;
Def: Specifier DecList error => Error type B: Missing semicolon;
Exp: FAULT error => Error type A: unknown lexeme;
   | ID LP Args error => error type B: Missing closing parenthesis;
   | Exp FAULT Exp error => Error type A: unknown lexeme;
```

Part 3 Bonus

1. single-line & multi-line comment

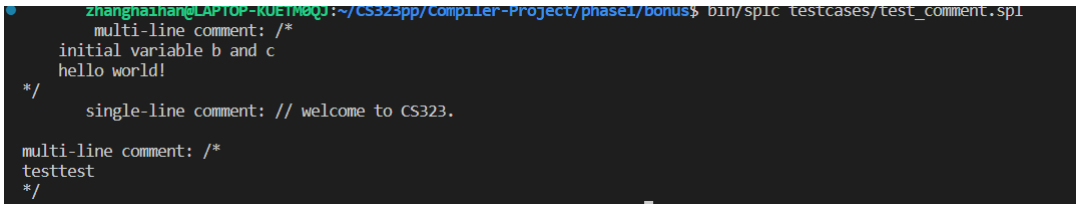
We can identify single-line and multi-line comments in the SPL file and output them on the terminal.

We use this code to match comments in `flex.l`.

```
"//"[^\n]*\n { printf("single-line comment: %s\n",yytext);}
"/*"([^*]|\*+[^/*])*"*/" { printf("multi-line comment: %s\n",yytext);}
```

And the test case is:

```
int main()
{
    int a = 1;
/*
    initial variable b and c
    hello world!
*/
    int b = 2;
// welcome to CS323.
/*
testtest
*/
}
```



2. macro preprocessor & file inclusion

We support the `#define` and `#include` sentences in the SPL file. And we can add them to the parsing tree.

In `flex.l`, we define two new tokens: `INCLUSION` and `DEFINE`, then use regular expressions to match:

```
#include[ ]+((<.*>)|(".*")) {return INCLUSION;}
#define[ ]+.+[ ]+.+  {return DEFINE;}
```

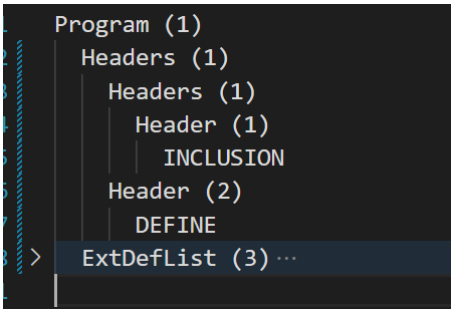
Meanwhile, we modify a few productions to restrict the `define` and `include` sentences only on the top of a SPL file: (Ignoring the action of each production)

```
Program : Headers ExtDefList ;
Headers : Headers Header ;
Header : INCLUSION | DEFINE ;
```

A simple test case is:

```
#include <stdio.h>
#define PI 3.14
int main(){
    int a = 1;
}
```

The output parsing tree is:



The output meets our expectations.

3. for statement

We set a token `FOR` in `flex.l` and add a production of `Stmt` in `syntax.y`: `FOR LP Exp SEMI Exp SEMI Exp RP Stmt` . The following shows the test case and result:

```
int main(){
    int i;
    for(i = 1; i < 5; i = i+1){
        printf("hello");
    }
}
```

```
StmtList (3)
  Stmt (3)
    FOR
    LP
    Exp (3)
      Exp (3)
        ID: i
      ASSIGN
      Exp (3)
        INT: 1
    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)
      CompSt (3)
        LC
        StmtList (4)
```