1. 学习flex&bison 书籍上的bison部分， 书上的例子都跑一边。

2. 写flex的代码

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "kaleidoscope.tab.h"

#define MAX_IDENTIFIER_LENGTH 100
%}

%option noyywrap

%{
// 符号表结构
struct Symbol {
    char name[MAX_IDENTIFIER_LENGTH];
    double value;
    struct Symbol* next;
};

struct Symbol* symbol_table = NULL; // 符号表头指针
%}

%%
[a-zA-Z][a-zA-Z0-9]* {
    yylval.identifier = strdup(yytext);
    return IDENTIFIER;
}

[0-9]+(\.[0-9]+)? {
    yylval.number = atof(yytext);
    return NUMBER;
}

[ \t]        ; // 忽略空格和制表符

[\n]         { return EOL; }

.            { return yytext[0]; }
%%

int yywrap() {
```

```
    return 1;
}
```

3. 写bison部分的代码

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "kaleidoscope.tab.h"

struct Symbol {
    char name[MAX_IDENTIFIER_LENGTH];
    double value;
    struct Symbol* next;
};

extern struct Symbol* symbol_table;

struct ASTNode {
    char* identifier;
    double number;
    struct ASTNode* left;
    struct ASTNode* right;
};

struct ASTNode* create_identifier_node(char* identifier) {
    struct ASTNode* node = (struct ASTNode*)malloc(sizeof(struct ASTNode));
    node->identifier = identifier;
    node->left = NULL;
    node->right = NULL;
    return node;
}

struct ASTNode* create_number_node(double number) {
    struct ASTNode* node = (struct ASTNode*)malloc(sizeof(struct ASTNode));
    node->number = number;
    node->left = NULL;
    node->right = NULL;
    return node;
}
```

```
37
38 void print_symbol_table() {
39     struct Symbol* symbol = symbol_table;
40     printf("Symbol Table:\n");
41     while (symbol != NULL) {
42         printf("  %s = %g\n", symbol->name, symbol->value);
43         symbol = symbol->next;
44     }
45 }
46
47 void yyerror(const char* msg) {
48     printf("语法错误: %s\n", msg);
49 }
50 %}
51
52 %union {
53     char* identifier;
54     double number;
55     struct ASTNode* node;
56 }
57
58 %token <identifier> IDENTIFIER
59 %token <number> NUMBER
60 %token EOL
61
62 %left '+' '-'
63 %left '*' '/'
64
65 %type <node> expr identifier_expr function_definition declaration
   arg_list
66
67 %start program
68
69 %%
70
71 program:
72     | program definition EOL { print_symbol_table(); }
73     ;
74
75 definition:
76     function_definition { /* do nothing */ }
77     | declaration { /* do nothing */ }
78     ;
```

```
79
80  function_definition:
81    IDENTIFIER '(' arg_list ')' '{' expr '}' {
82        struct Symbol* symbol = (struct Symbol*)malloc(sizeof(struct
    Symbol));
83        strcpy
84        (symbol->name, $1);
85        symbol->value = $6->number;
86        symbol->next = symbol_table;
87        symbol_table = symbol;
88        $$ = create_identifier_node($1);
89    }
90    ;
91
92  declaration:
93    IDENTIFIER '=' expr {
94        struct Symbol* symbol = (struct Symbol*)malloc(sizeof(struct
    Symbol));
95        strcpy(symbol->name, $1);
96        symbol->value = $3->number;
97        symbol->next = symbol_table;
98        symbol_table = symbol;
99        $$ = create_identifier_node($1);
100   }
101   ;
102
103 arg_list:
104   /* empty */
105   | IDENTIFIER { $$ = create_identifier_node($1); }
106   | arg_list ',' IDENTIFIER {
107       struct ASTNode* temp = create_identifier_node($3);
108       temp->left = $1;
109       $$ = temp;
110   }
111   ;
112
113 expr:
114   identifier_expr { $$ = $1; }
115   | NUMBER { $$ = create_number_node($1); }
116   | expr '+' expr {
117       struct ASTNode* node = create_identifier_node("+");
118       node->left = $1;
119       node->right = $3;
```

```
        $$ = node;
    }
    | expr '-' expr {
        struct ASTNode* node = create_identifier_node("-");
        node->left = $1;
        node->right = $3;
        $$ = node;
    }
    | expr '*' expr {
        struct ASTNode* node = create_identifier_node("*");
        node->left = $1;
        node->right = $3;
        $$ = node;
    }
    | expr '/' expr {
        struct ASTNode* node = create_identifier_node("/");
        node->left = $1;
        node->right = $3;
        $$ = node;
    }
    | '(' expr ')' { $$ = $2; }
    ;

identifier_expr:
    IDENTIFIER {
        struct Symbol* symbol = symbol_table;
        while (symbol != NULL) {
            if (strcmp(symbol->name, $1) == 0) {
                $$ = create_number_node(symbol->value);
                return;
            }
            symbol = symbol->next;
        }
        printf("变量 %s 未定义\n", $1);
        exit(1);
    }
    ;

%%

int main() {
    yyparse();
    return 0;
```

```
163 }
164
```

4. 编译

```
(base) ts@ml:~/bupt/flex_bison_book/kakeidocope2$ ls
kaleidoscope.l  kaleidoscope.tab.c  kaleidoscope.tab.h  kaleidoscope.y  lex.yy.c
(base) ts@ml:~/bupt/flex_bison_book/kakeidocope2$
```