

编译专题实验报告

实验三：语法分析

计算机2101 陈实

完成模式：独立完成

实验平台

1. 操作系统: WSL2 Ubuntu 20.04
2. 辅助工具: LEX 、 BISON

实验目的

1. 完成计算器
 - 支持加减乘除、括号、“-”、关系运算等操作;
 - 考虑小数
 - 考虑算符结合和优先级;
2. 输出抽象语法树AST

```
> ~/bianyi/lab3.1
> ./calc
10.5+5*(6-4)
ADDOP:
  NUMBER: 10.500000
  MULOP:
    NUMBER: 5.000000
  ADDOP:
    NUMBER: 6.000000
    NUMBER: 4.000000
= 20.500000
^C

> ~/bianyi/lab3.1
> date
Wed May 15 19:44:31 CST 2024
```

计算机2101 陈实
完成时间5月15日
独立完成

实验内容

1. lex文件

```
1 %option noyywrap
2
3 %{
4
5 #include "calc.tab.h"
6 %}
7
8 %%
9 \( { return LPAREN; }
10 \) { return RPAREN; }
11 "+"|"-" { yylval.op = yytext[0]; return ADDOP; }
12 "*"|"/" { yylval.op = yytext[0]; return MULOP; }
13 "|"|"&" { yylval.op = yytext[0]; return LOGOP; }
14 "%" {yylval.op = yytext[0]; return MODOP; }
15 [0-9]+|[0-9]+\.[0-9]*|[0-9]*\.[0-9]+ { yylval.num = atof(yytext);
  return NUMBER; }
```

```

16  " |\t { }
17  \r\n|\n|\r { return RET; }
18  %%%

```

1. 定义了括号、加减乘除、逻辑运算符、取余、数字、空格、换行符等词法单元;
2. 增加了stdlib.h的包含, 以便使用atof函数将字符串转换为浮点数;
3. 每个正则表达式对应一个返回值 (如LPAREN, RPAREN, ADDOP等), 这些返回值在calc.tab.h中定义;
4. 处理空格和制表符时忽略它们, 不产生任何动作;
5. 处理换行符时返回RET, 表明一行结束。

2. Bison文件

```

1  /* calc.y */
2  %{
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include "ast.h"
7
8  int yylex(void);
9  void yyerror(const char *s);
10
11 %}
12
13 %token RET
14 %token <num> NUMBER
15 %token <op> ADDOP MULOP LPAREN RPAREN LOGOP MODOP
16 %type <node> top line expr term factor
17
18 %start top
19
20 %union {
21     char    op;
22     double  num;
23     struct ASTNode *node;
24 }
25
26 %%
27
28 top
29 : top line {}
30 | {}
31
32 line
33 : expr RET
34 {
35     printAST($1, 0);
36     printf(" = %f\n", $1->data.num);
37     freeAST($1);
38 }
39
40 expr
41 : term

```

```

42 {
43     $$ = $1;
44 }
45 | expr ADDOP term
46 {
47     $$ = createNode("ADDOP", 0, $2, $1, $3);
48     $$→data.num = ($2 == '+') ? $1→data.num + $3→data.num : $1-
>data.num - $3→data.num;
49 }
50 | expr LOGOP term
51 {
52     $$ = createNode("LOGOP", 0, $2, $1, $3);
53     $$→data.num = ($2 == '|') ? $1→data.num || $3→data.num : $1-
>data.num && $3→data.num;
54 }
55 | expr MODOP term
56 {
57     $$ = createNode("MODOP", 0, '%', $1, $3);
58     $$→data.num = (int)$1→data.num % (int)$3→data.num;
59 }
60
61 term
62 : factor
63 {
64     $$ = $1;
65 }
66 | term MULOP factor
67 {
68     $$ = createNode("MULOP", 0, $2, $1, $3);
69     $$→data.num = ($2 == '*') ? $1→data.num * $3→data.num : $1-
>data.num / $3→data.num;
70 }
71
72 factor
73 : LPAREN expr RPAREN
74 {
75     $$ = $2;
76 }
77 | ADDOP factor
78 {
79     $$ = createNode("NEGATE", 0, $1, NULL, $2);
80     $$→data.num = ($1 == '-') ? -$2→data.num : $2→data.num;
81 }
82 | NUMBER
83 {
84     $$ = createNode("NUMBER", $1, 0, NULL, NULL);
85 }
86
87 %%
88
89 ASTNode *createNode(char *type, double num, char op, ASTNode *left,
ASTNode *right) {
90     ASTNode *node = (ASTNode *)malloc(sizeof(ASTNode));
91     node→type = strdup(type);

```

```

92     if (strcmp(type, "NUMBER") == 0) {
93         node->data.num = num;
94     } else {
95         node->data.op = op;
96     }
97     node->left = left;
98     node->right = right;
99     return node;
100 }
101
102 void printAST(ASTNode *node, int level) {
103     if (!node) return;
104     for (int i = 0; i < level; i++) printf(" ");
105     if (strcmp(node->type, "NUMBER") == 0) {
106         printf("%s: %f\n", node->type, node->data.num);
107     } else {
108         printf("%s: %c\n", node->type, node->data.op);
109     }
110     printAST(node->left, level + 1);
111     printAST(node->right, level + 1);
112 }
113
114 void freeAST(ASTNode *node) {
115     if (!node) return;
116     free(node->type);
117     freeAST(node->left);
118     freeAST(node->right);
119     free(node);
120 }
121
122 void yyerror(const char *s)
123 {
124     fprintf(stderr, "%s\n", s);
125 }
126
127 int main()
128 {
129     yyparse();
130     return 0;
131 }

```

1. 引入所需头文件:

- 引入了stdio.h、stdlib.h、string.h和自定义的ast.h, 以支持输入输出、内存管理和字符串操作。
- 声明了词法分析函数yylex和错误处理函数yyerror。

2. 定义词法单元和语法单元的类型:

- 词法单元包括: RET (回车)、NUMBER (数字)、ADDOP (加减操作符)、MULOP (乘除操作符)、LPAREN (左括号)、RPAREN (右括号)、LOGOP (逻辑操作符)和MODOP (取余操作符)。
- 定义了语法单元的类型 (top、line、expr、term、factor), 这些语法单元将使用抽象语法树节点 (ASTNode) 来表示。

3. 定义起始规则和联合体:

- top被定义为起始规则。
- 定义了%union来表示符号值的不同类型，包括字符（操作符）、双精度浮点数（数字）和抽象语法树节点（ASTNode）。

4. 语法规则及其动作：

- top规则：表示多个line的组合或为空，形成了程序的顶层结构。
- line规则：每行由一个表达式加换行符组成，处理表达式并输出结果，同时打印并释放抽象语法树。
- expr规则：定义了表达式的结构，可以是一个term、ADDOP运算、LOGOP运算或MODOP运算。相应的动作创建AST节点并计算结果。
- term规则：定义了项的结构，可以是一个factor或MULOP运算，动作创建AST节点并计算结果。
- factor规则：定义了因子的结构，可以是括号中的表达式、带符号的因子或数字，动作创建AST节点并处理数值。

5. 辅助函数：

- createNode：创建一个AST节点，初始化类型、数据和子节点。
- printAST：递归打印AST，用于调试和输出树的结构。
- freeAST：递归释放AST节点的内存，防止内存泄漏。
- yyerror：错误处理函数，打印错误信息。
- main函数：启动解析过程。

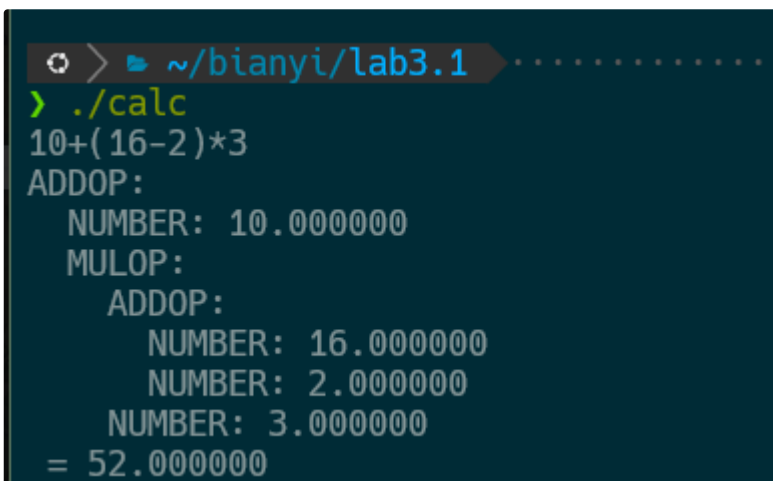
3. makefile文件

```
1  calc: calc.y calc.l ast.h
2      bison -d calc.y
3      flex calc.l
4      gcc -o $@ calc.tab.c lex.yy.c -lfl -lm
5
6  clean:
7      rm -f calc.tab.* lex.yy.c calc
```

1. 使用bison和flex生成对应的C文件，然后编译链接生成可执行文件calc。
2. clean规则用于清理生成的文件。

实验结果

1. $10+(16-2)*3$



```
> ~/bianyi/lab3.1
> ./calc
10+(16-2)*3
ADDOP:
  NUMBER: 10.000000
MULOP:
  ADDOP:
    NUMBER: 16.000000
    NUMBER: 2.000000
    NUMBER: 3.000000
= 52.000000
```

2. $3.14 \times 2.718 + 1.618$

```
> ~/bianyi/lab3.1
> ./calc
3.14*5-3
ADDOP: g
MULOP: g
NUMBER: 3.140000
NUMBER: 5.000000
NUMBER: 3.000000
= 12.700000
```

实验总结

1. 通过本次实验，我学会了使用Bison和Flex工具来实现语法分析，实现了一个简单的计算器，支持加减乘除、括号、小数、逻辑运算和取余等操作。
2. 通过实现抽象语法树AST，我更好地理解了语法分析的过程，掌握了如何将表达式转换为树形结构，并实现计算和输出。
3. 通过调试和测试，我发现了一些问题，如优先级和结合性的处理、负号的处理等，通过修改代码和调整规则，最终实现了正确的计算和输出。