

constant.c

```
1 #include "constant.h"
2 #include<stdbool.h>
3 #include<string.h>
4 #include<stdio.h>
5 #include <stdlib.h>
6 #include<stdlib.h>
7 #include<ctype.h>
8
9 int COND_OPERATORS_SIZE = 6;
10 char COND_OPERATORS[6][5] = {
11     "lt", "gt", "eq", "neq", "le", "ge"
12 };
13
14 int ARITHMATIC_OPERATORS_SIZE = 6;
15 char ARITHMATIC_OPERATORS[6][5] = {
16     "add", "sub", "mul", "div", "dif", "rem"
17 };
18
19 bool isArithOp(char *ch){
20     for(int i=0; i<ARITHMATIC_OPERATORS_SIZE; i++){
21         if( strcmp(ARITHMATIC_OPERATORS[i], ch) == 0 ) return true;
22     }
23     return false;
24 }
25
26 bool isCondOpValid(char* op){
27     //char arr[6][5] = { "lt", "gt", "eq", "neq", "le", "ge" };
28     for(int i=0; i<COND_OPERATORS_SIZE; i++){
29         if(strcmp(COND_OPERATORS[i],op) == 0) return true;
30     }
31     return false;
32 }
33
34 bool isConditionValid(double left, char* op, double right){
35     if( strcmp(op,"lt",2) == 0) return (left < right);
36     if( strcmp(op,"gt",2) == 0) return (left > right);
37     if( strcmp(op,"eq",2) == 0) return (left == right);
38     if( strcmp(op,"neq",2) == 0) return (left != right);
39
40     if( strcmp(op,"le",2) == 0) return (left <= right);
41     if( strcmp(op,"ge",2) == 0) return (left >= right);
42     if( strcmp(op,"neq",2) == 0) return (left != right); // 2 fine also
43 }
44
45 double getValue(double left, double right, char *op){
46     // "add", "sub"
47     if( strcmp(op,ARITHMATIC_OPERATORS[0],3) == 0 ) {
48         printf("\nadd %lf %lf\n",left,right);
49         return left+right;
50     }
51     if( strcmp(op,ARITHMATIC_OPERATORS[1],3) == 0 ) return left-right;
52
53     // "mul", "div"
54     if( strcmp(op,ARITHMATIC_OPERATORS[2],3) == 0 ) return left*right;
55     if( strcmp(op,ARITHMATIC_OPERATORS[3],3) == 0 ) return left/right;
56
57     // "dif", "rem"
58     if( strcmp(op,ARITHMATIC_OPERATORS[4],3) == 0 ) return left>right ? left-right : right-
59     left;
60     if( strcmp(op,ARITHMATIC_OPERATORS[5],3) == 0 ) return ((int)left) % ((int)right);
61
62 }
63
64 char* trim(char *s) {
65     // Trim trailing whitespace
66     char *end = s + strlen(s) - 1;
67     while(end > s && isspace((unsigned char)*end)) {
68         end--;
69     }
```

```
70 *(end + 1) = '\0';
71
72 // Trim leading whitespace
73 while(*s && isspace(((unsigned char)*s)) {
74     s++;
75 }
76
77 return s;
78 }
79
80 void processStatement(const char *stmt, char *leftVar, char *varBeforeOp, char *operator,
81 char *varAfterOp) {
82     char buffer[100];
83     int len = strlen(stmt);
84     int bufPos = 0;
85     int state = 0;
86     // 0: searching for leftVar,
87     // 1: searching for operator,
88     // 2: searching for varAfterOp
89     for (int i = 0; i < len; i++) {
90         char ch = stmt[i];
91
92         // Skip spaces
93         if (isspace(ch) || ch == '=') {
94             if (bufPos > 0) {
95                 buffer[bufPos] = '\0';
96                 switch (state) {
97                     case 0:
98                         strcpy(leftVar, buffer);
99                         state++;
100                         break;
101                     case 1:
102                         if( isArithOp(buffer) ){
103                             strcpy(operator, buffer);
104                             state++;
105                         }
106                     else {
107                         strcpy(varBeforeOp, buffer);
108                     }
109                     break;
110                 case 2:
111                     strcpy(varAfterOp, buffer);
112                     state++;
113                     break;
114                 }
115                 bufPos = 0;
116             }
117             continue;
118         }
119         if (isalnum(ch) || ch == '.' ) {
120             buffer[bufPos++] = ch;
121         }
122     }
123
124
125     // Handle any remaining items in the buffer
126     if (bufPos > 0) {
127         buffer[bufPos] = '\0';
128         if(state == 2){
129             strcpy(varAfterOp, buffer);
130         }
131     }
132
133     //printf("leftVar:~%s-, varBeforeOp: ~%s-, operator:~%s-, varAfterOp: ~%s-\n", leftVar,
134     varBeforeOp, operator, varAfterOp);
135 }
```