

var_list.c

```
1  #include<stdio.h>
2  #include <stddef.h>
3  #include<string.h>
4  #include<stdlib.h>
5  #include "var_list.h"
6  #include <stdbool.h>
7
8  const int KEYS_SIZE = 25;
9  char KEYS[50][25] = {
10     "void", "int","double","float","justInCase",
11     "println","discard","till","import",
12     "static","void","entryPoint",
13     "lt","gt","eq","neq","le","ge",
14     "add", "sub", "mul", "div", "dif", "rem"
15 };
16
17 struct VARIABLE *head = NULL;
18 struct VARIABLE *tail = NULL;
19
20 struct VARIABLE* createNode(const char *name, char *type, double value) {
21     struct VARIABLE *newNode = (struct VARIABLE*) malloc(sizeof(struct VARIABLE));
22     if(!newNode) {
23         printf("Memory allocation failed.\n");
24         return NULL;
25     }
26
27     strncpy(newNode->name, name, sizeof(newNode->name) - 1);
28     newNode->value = value;
29     strncpy(newNode->type,type,sizeof(newNode->type)-1);
30     newNode->prev = NULL;
31     newNode->next = NULL;
32     return newNode;
33 }
34
35 void insertVariable(char *name, char *type, double val) {
36
37     for(int i=0; i<KEYS_SIZE; i++){
38         if(strncmp(KEYS[i],name) == 0){
39             printf("Keyword '%s' can't be variable\n",name);
40             return;
41         }
42     }
43
44     struct VARIABLE *var = createNode(name,type,val);
45
46     if( strncmp("int",type,3) == 0 ){ // integer
47         val = (double)( (int)val ); // ignoring after decimal
48     }
49
50     if(tail == NULL) {
51         head = var;
52         tail = var;
53     }
```

```

54     else {
55         var->prev = tail;
56         tail->next = var;
57         tail = var;
58     }
59 }
60
61 int getTotalVar(){
62     struct VARIABLE *ptr;
63     int count=0;
64     ptr = head;
65     while(ptr != NULL){
66         count++;
67         ptr = ptr->next;
68     }
69     return count;
70 }
71
72 void updateVariable(char *name, double val){
73
74     struct VARIABLE *ptr;
75     ptr = head;
76     while(ptr != NULL){
77         if( strcmp(ptr->name,name) == 0 ){
78
79             if( strncmp("int",ptr->type,3) == 0 ) val = (int)val;
80
81             ptr->value = val;
82             break;
83         }
84         ptr = ptr->next;
85     }
86 }
87
88 void deleteVariable(char *name){
89     struct VARIABLE *ptr;
90     ptr = head;
91     while(ptr != NULL){
92         if( strcmp(ptr->name, name) == 0 ){
93             // first delete
94             if(ptr == head){
95                 //single node
96                 if(ptr == tail) tail = NULL;
97                 head = ptr->next;
98             }
99             else if(ptr == tail){ // last delete
100                 tail = tail->prev;
101                 tail->next = NULL;
102             }
103             else{
104                 ptr->prev->next = ptr->next;
105                 ptr->next->prev = ptr->prev;
106             }
107         }
108         ptr = ptr->next;
109     }

```

```

110 }
111
112 bool doesVariableExists(char *name){
113     struct VARIABLE *ptr = head;
114
115     while (ptr != NULL)
116     {
117         if( strcmp(name,ptr->name) == 0 ) return true;
118         ptr = ptr->next;
119     }
120     return false;
121 }
122
123 struct VARIABLE* getVariable(char* name){
124     //printf("printing from inner\n");
125     //printAll();
126     struct VARIABLE *ptr = head;
127     while (ptr != NULL)
128     {
129         //printf("(%s %ld),",ptr->name,ptr->value);
130         if( strcmp(name,ptr->name) == 0 ){
131             //printf("\n value returning %s %lf\n",ptr->name,ptr->value);
132             return ptr;
133         }
134         ptr = ptr->next;
135     }
136     return NULL;
137 }
138
139 double getValueOrDefault(char* name){
140     struct VARIABLE *ptr = head;
141     while (ptr != NULL)
142     {
143         if( strcmp(name,ptr->name) == 0 ){
144             return ptr->value;
145         }
146         ptr = ptr->next;
147     }
148     return 0;
149 }
150
151 char* getFormattedValueOrDefault(char *name){
152     struct VARIABLE* var = getVariable(name);
153     if(var == NULL){
154         return "0";
155     }
156
157     char *arr = (char *) malloc(20);
158
159     if ( strcmp(var->type,"int") == 0){
160         int num = (int)(var->value);
161         sprintf(arr, "%d", num);
162         return arr;
163     }
164
165     double num = (var->value);

```

```

166     sprintf(arr, "%lf", num);
167     return arr;
168 }
169
170 void printAll(){
171     printf("\n");
172     if(head == NULL) return;
173
174
175     printf("-----Printing all variables-----\n");
176
177     struct VARIABLE *ptr;
178     ptr = head;
179     while(ptr != NULL){
180         if ( strcmp(ptr->type,"int") == 0){
181             printf("%s(%s) %d -> ",ptr->name,ptr->type,(int)ptr->value);
182         }
183         else{
184             printf("%s(%s) %lf -> ",ptr->name,ptr->type,ptr->value);
185         }
186         ptr = ptr->next;
187     }
188     printf("\n\n");
189 }
190
191
192 bool *validityList = NULL;
193 int validIndex = 0;
194
195 void pushValidity(bool val){
196     validIndex++;
197     validityList = (bool*) realloc(validityList,validIndex);
198     validityList[validIndex-1] = val;
199 }
200
201 bool getCurrentValidity(){
202     if(validIndex <= 0) return true;
203     return validityList[validIndex-1];
204 }
205
206 bool popValidity(){
207     bool val = validityList[validIndex];
208     validIndex--;
209     return val;
210 }
211

```