

prototype_list.c

```

1  #include<stdio.h>
2  #include <stddef.h>
3  #include<string.h>
4  #include<stdlib.h>
5  #include<math.h>
6  #include "prototype_list.h"
7  #include <stdbool.h>
8
9  struct PROTOTYPE *libraryProtoHead = NULL;
10 struct PROTOTYPE *libraryProtoTail = NULL;
11
12 struct PROTOTYPE *protoHead = NULL;
13 struct PROTOTYPE *protoTail = NULL;
14
15 char **imports = 0;
16 int totalImport = 0;
17
18 void insertImport(char fullImp[20]){
19     char *imp = malloc(14*sizeof(char));
20     strncpy(imp,fullImp+7, strlen(fullImp) - 8 );
21
22     if( isImportImported(imp) ){
23         printf("\nImport already exists\n");
24         return;
25     }
26
27     totalImport++;
28     imports = (char **) realloc( imports, totalImport * 20 * sizeof(char) );
29     imports[totalImport-1] = calloc(20, sizeof(char));
30
31     strcpy(imports[totalImport-1],imp);
32     printf("\nImported %s\n",imp);
33     free(imp);
34 }
35
36 bool isImportImported(char *imp){
37     for(int i=0; i<totalImport; i++){
38         if( strcmp(imports[i], imp) == 0) return true;
39     }
40     return false;
41 }

```

```

45 void printAllImports(){
46     printf("\nPrinting all imports:\n");
47     for(int i=0; i<totalImport; i++){
48         printf("%s\n",imports[i]);
49     }
50 }
51
52 struct PARAMETER* createParameter(const char *type, double value) {
53     struct PARAMETER *node = (struct PARAMETER*) malloc(sizeof(struct
54     PARAMETER));
55     if(!node) { printf("Memory allocation failed.\n"); return NULL; }
56
57     strncpy(node->type, type, sizeof(node->type) - 1);
58     node->value = value;
59
60     node->prev = NULL;
61     node->next = NULL;
62     return node;
63 }
64
65 void insertParameter( struct PARAMETER **head, struct PARAMETER **tail, char
66 *type, double val){
67
68     struct PARAMETER *var = createParameter(type,val);
69
70     if(*tail == NULL) {
71         *head = var;
72         *tail = var;
73     }
74     else {
75         var->prev = (*tail);
76         (*tail)->next = var;
77         (*tail) = var;
78     }
79 }
80
81 struct PROTOTYPE* createProto(char *type, char *name, char *libraryName,
82 struct PARAMETER *paramsHead, struct PARAMETER *paramsTail ) {
83
84     struct PROTOTYPE *newNode = (struct PROTOTYPE*) malloc(sizeof(struct
85     PROTOTYPE));
86     if(!newNode) {

```

```

89     printf("Memory allocation failed.\n");
90     return NULL;
91 }
92
93 //printf("Inside-3 %s \n", type);
94 strncpy(newNode->funcType, type, sizeof(newNode->funcType) - 1);
95
96 //printf("Inside-4\n");
97 strncpy(newNode->funcName, name, sizeof(newNode->funcName)-1);
98
99 strncpy(newNode->libraryName, libraryName, sizeof(newNode->libraryName)
-1);
100
101 //printf("Inside-5\n");
102 newNode->paramsHead = paramsHead;
103 newNode->paramsTail = paramsTail;
104
105 newNode->prev = NULL;
106 newNode->next = NULL;
107
108 //printf("Inside - 6\n");
109
110 // printf("inside create: ");
111 // printProto(newNode);
112
113     return newNode;
114 }
115
116 void insertLibraryProto(struct PROTOTYPE* var) {
117
118     if(libraryProtoTail == NULL) {
119         libraryProtoHead = var;
120         libraryProtoTail = var;
121     }
122     else {
123         var->prev = libraryProtoTail;
124         libraryProtoTail->next = var;
125         libraryProtoTail = var;
126     }
127 }
128
129 void printAllLibraryFunction(){
130     printf("\nAll library functions are:\n");
131     struct PROTOTYPE *ptr = libraryProtoHead;
132
133     while( ptr != NULL ){
134         printProto(ptr, false);
135
136         ptr = ptr->next;
137     }
138
139     void insertProto(struct PROTOTYPE* var) {
140
141         if(protoTail == NULL) {
142             protoHead = var;
143             protoTail = var;
144         }
145         else {
146             var->prev = protoTail;
147             protoTail->next = var;
148             protoTail = var;
149         }
150     }
151
152     struct PROTOTYPE* getOriginalProto(struct PROTOTYPE* proto, bool isLibrary){
153
154         struct PROTOTYPE *ptr = isLibrary ? libraryProtoHead : protoHead;
155         while (ptr != NULL)
156         {
157             bool isNameSame = strcmp( ptr->funcName ,ptr->funcName) == 0 ?
true : false;
158
159             if(isNameSame){
160
161                 struct PARAMETER *param1 = proto->paramsHead;
162                 struct PARAMETER *param2 = ptr->paramsTail;
163
164                 while ( param1 != NULL && param2 != NULL )
165                 {
166                     if( strcmp(param2->type, "any") == 0) {}
167                     else if( strcmp(param1->type, param2->type) != 0){
168                         break;
169                     }
170
171                     param1 = param1->next;
172                     param2 = param2->prev;
173                 }
174
175                 if(param1 == NULL && param2 == NULL){
176                     return ptr;
177                 }
178
179             }
180

```

```

181         ptr = ptr->next;
182     }
183     return NULL;
184 }
185
186 bool doesProtoExists(struct PROTOTYPE* proto, bool isLibrary){
187     return getOriginalProto(proto,isLibrary) != NULL;
188 }
189
190 void printAllProto(){
191     struct PROTOTYPE *ptr = protoHead;
192     while (ptr != NULL)
193     {
194         printProto(ptr,true);
195         ptr = ptr->next;
196     }
197 }
198
199 // prints data also if reverse is false
200 void printProto(struct PROTOTYPE *ptr, bool reverse){
201     printf("%s %s(",ptr->funcType, ptr->funcName);
202
203     struct PARAMETER* param = reverse ? ptr->paramsTail : ptr->paramsHead;
204     while(param != NULL){
205         printf( "%s", param->type);
206         //if(!reverse) printf("%lf",param->value);
207         if( (reverse ? param->prev : param->next) != NULL){ printf( " , "); }
208         param = reverse ? param->prev : param->next;
209     }
210     printf("\n");
211 }
212
213 double getLibrayFunctionResult(char* name, struct PARAMETER* params){
214     if( strcmp(name, "max") == 0 ){
215         return fmax(params->value, params->next->value);
216     }
217 }
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
if( strcmp(name, "sqrt") == 0 ){
    if(params->value < 0){
        printf("\nStay in real world\n");
        return 0;
    }
    return sqrt(params->value);
}

if( strcmp(name, "scanInt") == 0 ){
    printf("\nTaking int input 0\n");
    return 0;
}

if( strcmp(name, "scan") == 0 ){
    printf("\nTaking input 0.0\n");
    return 0;
}

if( strcmp(name, "toInt") == 0 ){
    return (int)(params->value);
}

if( strcmp(name, "toFloat") == 0 ){
    return (float)(params->value);
}

if( strcmp(name, "toDouble") == 0 ){
    return (double)(params->value);
}

if( strcmp(name, "show") == 0 ){
    printf("From show function: %lf \n", params->value);
    return 0;
}

return 0;
}

void initializeLibraryFunction(){
    struct PARAMETER *head = NULL, *tail = NULL;

    //scanInt stdio
    {
        head = NULL; tail = NULL;

```

```

275 struct PROTOTYPE* scanInt = createProto("int", "scanInt", "stdio", head,
276 tail);
277 insertLibraryProto(scanInt);
278 }
279 //scan stdio
280 {
281     head = NULL; tail = NULL;
282     struct PROTOTYPE* scan = createProto("float", "scan", "stdio", head, tail);
283     insertLibraryProto(scan);
284 }
285
286 // show stdio
287 {
288     head = NULL; tail = NULL;
289     insertParameter(&head, &tail, "any", -1);
290     struct PROTOTYPE* show = createProto("void", "show", "stdio", head, tail);
291     insertLibraryProto(show);
292 }
293
294 // max math
295 {
296     head = NULL; tail = NULL;
297     insertParameter(&head, &tail, "any", -1);
298     insertParameter(&head, &tail, "any", -1);
299
300     struct PROTOTYPE* max = createProto("float", "max", "math", head, tail);
301     insertLibraryProto(max);
302 }
303
304
305 // sqrt math
306 {
307     head = NULL; tail = NULL;
308     insertParameter(&head, &tail, "any", -1);
309     struct PROTOTYPE* sqrt = createProto("double", "sqrt", "math", head, tail);
310     insertLibraryProto(sqrt);
311 }
312
313 //toInt converter
314 {
315     head = NULL; tail = NULL;
316     insertParameter(&head, &tail, "any", -1);
317     struct PROTOTYPE* toInt = createProto("int", "toInt", "converter", head,
318 tail);
319     insertLibraryProto(toInt);
320 }
321
322 //toFloat converter
323 {
324     head = NULL; tail = NULL;
325     insertParameter(&head, &tail, "any", -1);
326     struct PROTOTYPE* toFloat = createProto("float", "toFloat", "converter",
327 head, tail);
328     insertLibraryProto(toFloat);
329 }
330
331 //toDouble converter
332 {
333     head = NULL; tail = NULL;
334     insertParameter(&head, &tail, "any", -1);
335     struct PROTOTYPE* toDouble = createProto("double", "toDouble", "converter",
336 head, tail);
337     insertLibraryProto(toDouble);
338 }

```