prototype_list.c

```c
#include<stdio.h>
#include <stddef.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#include "prototype_list.h"
#include <stdbool.h>

struct PROTOTYPE *libraryProtoHead = NULL;
struct PROTOTYPE *libraryProtoTail = NULL;

struct PROTOTYPE *protoHead = NULL;
struct PROTOTYPE *protoTail = NULL;


char **imports = 0;
int totalImport = 0;


void insertImport(char fulImp[20]){
    char *imp = malloc(14*sizeof(char));

    strncpy(imp,fulImp+7, strlen(fulImp) - 8 );

    if( isImportImported(imp) ){
        printf("\nImport already exists\n");
        return;
    }

    totalImport++;
    imports = (char **) realloc( imports, totalImport * 20 * sizeof(char) );
    imports[totalImport-1] = calloc(20, sizeof(char));

    strcpy(imports[totalImport-1],imp);
    printf("\nImported %s\n",imp);
    free(imp);
}

bool isImportImported(char *imp){
    for(int i=0; i<totalImport; i++){
        if( strcmp(imports[i], imp)  == 0) return true;
    }
    return false;
}


void printAllImports(){
    printf("\nPrinting all imports:\n");
    for(int i=0; i<totalImport; i++){
        printf("%s\n",imports[i]);
    }
}

struct PARAMETER* createParameter(const char *type,double value) {

    struct PARAMETER *node = (struct PARAMETER*) malloc(sizeof(struct PARAMETER));

    if(!node) { printf("Memory allocation failed.\n"); return NULL; }

    strncpy(node->type, type, sizeof(node->type) - 1);
    node->value = value;

    node->prev = NULL;
    node->next = NULL;
    return node;
}


void insertParameter( struct PARAMETER **head, struct PARAMETER **tail, char *type, double val){

    struct PARAMETER *var = createParameter(type,val);

    if(*tail == NULL) {
        *head = var;
        *tail = var;
    }
    else {
        var->prev = (*tail);
        (*tail)->next = var;
        (*tail) = var;
    }
}


struct PROTOTYPE* createProto(char *type, char *name, char *libraryName,
        struct PARAMETER *paramsHead, struct PARAMETER *paramsTail ) {

    struct PROTOTYPE *newNode = (struct PROTOTYPE*) malloc(sizeof(struct PROTOTYPE));
    if(!newNode) {
```

```
        printf("Memory allocation failed.\n");
        return NULL;
    }

    //printf("Inside-3 %s \n",type);
    strncpy(newNode->funcType, type, sizeof(newNode->funcType) - 1);

    //printf("Inside-4\n");
    strncpy(newNode->funcName, name ,sizeof(newNode->funcName)-1);

    strncpy(newNode->libraryName, libraryName ,sizeof(newNode->libraryName)
-1);

    //printf("Inside-5\n");
    newNode->paramsHead = paramsHead;
    newNode->paramsTail = paramsTail;

    newNode->prev = NULL;
    newNode->next = NULL;

    //printf("Inside - 6\n");

    // printf("inside create: ");
    // printProto(newNode);

    return newNode;
}

void insertLibraryProto(struct PROTOTYPE* var) {

    if(libraryProtoTail == NULL) {
        libraryProtoHead = var;
        libraryProtoTail = var;
    }
    else {
        var->prev = libraryProtoTail;
        libraryProtoTail->next = var;
        libraryProtoTail = var;
    }
}

void printAllLibraryFunction(){
    printf("\nAll library functions are:\n");
    struct PROTOTYPE *ptr = libraryProtoHead;

    while( ptr != NULL ){
        printProto(ptr, false);
        ptr = ptr->next;
    }
}

void insertProto(struct PROTOTYPE* var) {

    if(protoTail == NULL) {
        protoHead = var;
        protoTail = var;
    }
    else {
        var->prev = protoTail;
        protoTail->next = var;
        protoTail = var;
    }
}


struct PROTOTYPE* getOriginalProto(struct PROTOTYPE* proto, bool isLibrary){

    struct PROTOTYPE *ptr = isLibrary ? libraryProtoHead : protoHead;
    while (ptr != NULL)
    {
        bool isNameSame = strcmp( proto->funcName ,ptr->funcName) == 0 ?
true : false;

        if(isNameSame){

            struct PARAMETER *param1 = proto->paramsHead;
            struct PARAMETER *param2 = ptr->paramsTail;

            while ( param1 != NULL && param2 != NULL )
            {
                if( strcmp(param2->type, "any") == 0) {}
                else if( strcmp(param1->type, param2->type)  != 0){
                    break;
                }

                param1 = param1->next;
                param2 = param2->prev;
            }

            if(param1 == NULL && param2 == NULL){
                return ptr;
            }

        }
```

```
        ptr = ptr->next;
    }
    return NULL;
}

bool doesProtoExists(struct PROTOTYPE* proto, bool isLibrary){
    return getOriginalProto(proto,isLibrary) != NULL;
}


void printAllProto(){
    struct PROTOTYPE *ptr = protoHead;

    while (ptr != NULL)
    {
        printProto(ptr,true);
        ptr = ptr->next;
    }
}

// prints data also if reverse is false
void printProto(struct PROTOTYPE *ptr, bool reverse){

    printf("%s %s(",ptr->funcType, ptr->funcName);

    struct PARAMETER* param = reverse ? ptr->paramsTail : ptr->paramsHead;

    while(param != NULL){
        printf("%s",param->type);

        //if(!reverse) printf("'%lf'",param->value);

        if( (reverse ? param->prev : param->next) != NULL){ printf(","); }

        param = reverse ? param->prev : param->next;
    }
    printf(")\n");

}

double getLibrayFunctionResult(char* name, struct PARAMETER* params){

    if( strcmp(name, "max") == 0 ){
        return fmax(params->value, params->next->value);
    }

    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;
```

```
    struct PROTOTYPE* scanInt = createProto("int","scanInt","stdio",head,
tail);
    insertLibraryProto(scanInt);
    }

    //scan stdio
    {
    head = NULL; tail = NULL;
    struct PROTOTYPE* scan = createProto("float","scan","stdio",head,tail);
    insertLibraryProto(scan);
    }

    // show stdio
    {
    head = NULL; tail = NULL;
    insertParameter(&head,&tail,"any",-1);
    struct PROTOTYPE* show = createProto("void","show","stdio",head,tail);
    insertLibraryProto(show);
    }

    // max math
    {
    head = NULL; tail = NULL;
    insertParameter(&head,&tail,"any",-1);
    insertParameter(&head,&tail,"any",-1);

    struct PROTOTYPE* max = createProto("float","max","math",head,tail);
    insertLibraryProto(max);
    }


    // sqrt math
    {
    head = NULL; tail = NULL;
    insertParameter(&head,&tail,"any",-1);
    struct PROTOTYPE* sqrt = createProto("double","sqrt","math",head,tail);
    insertLibraryProto(sqrt);
    }

    //toInt converter
    {
    head = NULL; tail = NULL;
    insertParameter(&head,&tail,"any",-1);
    struct PROTOTYPE* toInt = createProto("int","toInt","converter",head,
tail);
    insertLibraryProto(toInt);
    }


    //toFloat converter
    {
    head = NULL; tail = NULL;
    insertParameter(&head,&tail,"any",-1);
    struct PROTOTYPE* toFloat = createProto("float","toFloat","converter",
head,tail);
    insertLibraryProto(toFloat);
    }

    //toDouble converter
    {
    head = NULL; tail = NULL;
    insertParameter(&head,&tail,"any",-1);
    struct PROTOTYPE* toDouble = createProto("double","toDouble","converter"
,head,tail);
    insertLibraryProto(toDouble);
    }

}

```