

## Lab-4 system security

Praneesh R V

CB.SC.U4CYS23036

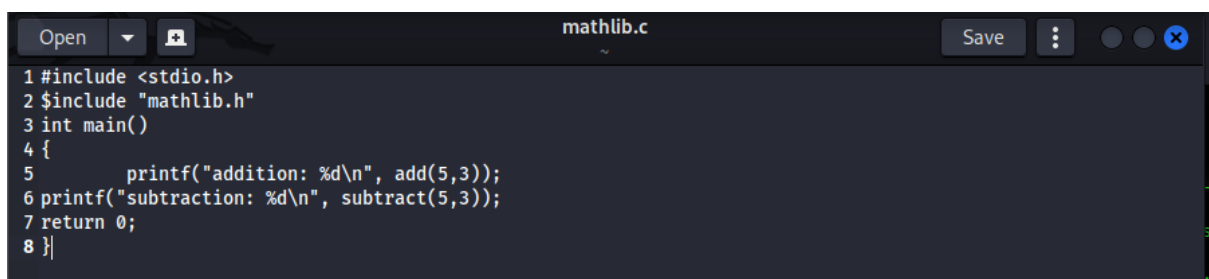
Creating static library:



The screenshot shows a code editor window titled 'mathlib.h'. The editor has a dark theme and a toolbar at the top with 'Open', 'Save', and window control buttons. The code is as follows:

```
1 #ifndef MATHLIB_H
2 #define MATHLIB_H
3
4 int add(int a, int b);
5 int subtract(int a, int b);
6 int multiply(int a, int b);
7 float divide(float a, float b);
8
9 #endif
```

The status bar at the bottom indicates 'C/ObjC Header', 'Tab Width: 8', 'Ln 9, Col 7', and 'INS'.



The screenshot shows a code editor window titled 'mathlib.c'. The editor has a dark theme and a toolbar at the top with 'Open', 'Save', and window control buttons. The code is as follows:

```
1 #include <stdio.h>
2 #include "mathlib.h"
3 int main()
4 {
5     printf("addition: %d\n", add(5,3));
6     printf("subtraction: %d\n", subtract(5,3));
7     return 0;
8 }
```

The status bar at the bottom indicates 'C/ObjC Source', 'Tab Width: 8', 'Ln 8, Col 1', and 'INS'.

```
(kali㉿kali)-[~]  
$ gcc -c mathlib.c -o mathlib.o  
  
(kali㉿kali)-[~]  
$ ar rcs libmathlib.a mathlib.o
```

```
(kali㉿kali)-[~]  
$ gcc -c -fPIC mathlib.c -o mathlib.o  
  
(kali㉿kali)-[~]  
$ gcc -shared -o libmathlib.so mathlib.o
```



```
Open  main.c  Save  [Menu]  [Close]  
1 #include "mathlib.h"  
2  
3 int add(int a, int b) { return a + b; }  
4 int subtract(int a, int b) { return a - b; }  
5 int multiply(int a, int b) { return a * b; }  
6 float divide(float a, float b) { return a / b; }
```

(gedit:799554): Gtk-WARNING \*\*: 06:58:09.468: Calling org.xfce.Session

```
(kali㉿kali)-[~]  
$ gcc main.c -L. -lmathlib -o static_program  
  
(kali㉿kali)-[~]  
$ █  
Home
```

```
(kali㉿kali)-[~]  
$ gcc main.c -L. -lmathlib -o dynamic_program  
  
(kali㉿kali)-[~]  
$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH  
  
(kali㉿kali)-[~]  
$ █  
Home
```

## Evaluation Questions

### 1. Theoretical Understanding

a Explain the key differences between static and dynamic linking

Ans.) Static Linking: The library code is copied into the executable at compile time.

Dynamic Linking: The library is loaded at runtime, enabling shared use across programs.

b. What are the advantages and disadvantages of each linking method?

Ans.) Static Linking Advantages: Independence from external files at runtime, simpler deployment.

Static Linking Disadvantages: Larger executable size, need for recompilation after library updates.

Dynamic Linking Advantages: Smaller executables, runtime flexibility, shared memory usage.

Dynamic Linking Disadvantages: Dependency issues, potential version conflicts.

c. How does the memory usage differ between static and dynamic libraries?

Ans.) Static libraries increase memory usage as each program includes its own copy of the library.

Dynamic libraries optimize memory by sharing the same library across multiple programs.

## **2. Technical Analysis**

a. Compare the file sizes of the executables created with static and dynamic linking

Ans.) Use `ls -lh` or `dir` to compare the file sizes of static and dynamically linked executables.

b. What happens if you try to run the dynamically linked program without setting `LD_LIBRARY_PATH`?

Ans.) The dynamically linked program will fail with a "library not found" error unless the library is in a standard path.

c. How would version conflicts arise in dynamic libraries, and how can they be managed?

Ans.) Conflicts arise if different programs require different versions of a shared library. These can be managed using tools like `ldconfig` or versioned library filenames (e.g., `libname.so.1`).

### **3. Practical Application**

a. When would you choose static linking over dynamic linking in a real-world project?

Ans.) Use static linking for standalone applications with no runtime dependencies. Dynamic linking is preferable for modular, large-scale systems.

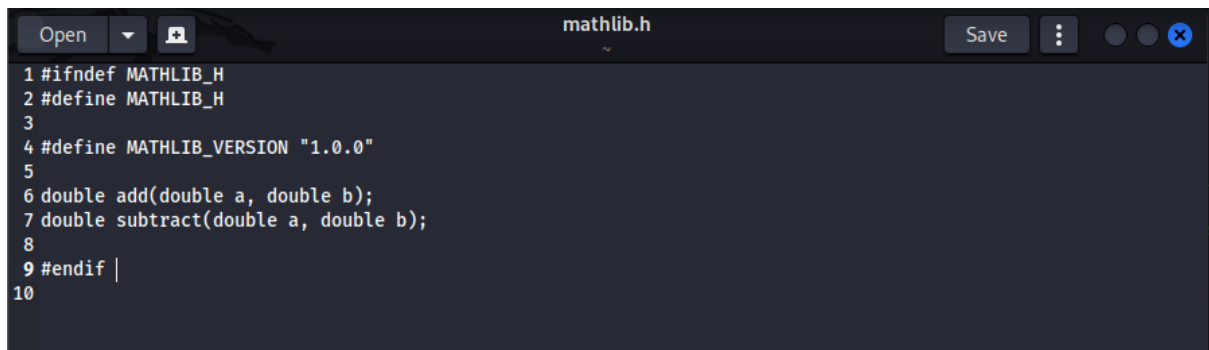
b. How does static linking affect application deployment compared to dynamic linking?

Ans.) Static linking simplifies deployment but increases the size of executables. Dynamic linking requires ensuring all required libraries are available at runtime.

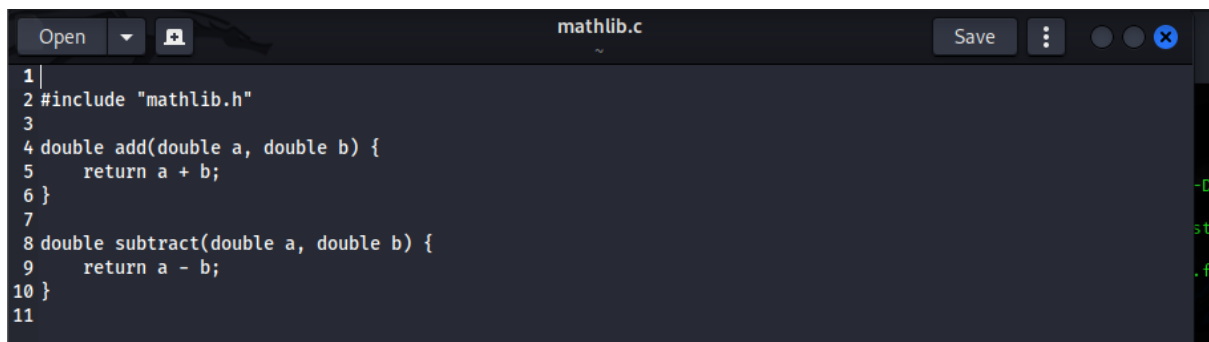
## Advanced Challenge Tasks

### 1. Library Version Management:

a.) Modify the math library to include version information

A screenshot of a code editor window titled 'mathlib.h'. The editor has a dark theme and a toolbar at the top with 'Open', 'Save', and window control buttons. The code is as follows:

```
1 #ifndef MATHLIB_H
2 #define MATHLIB_H
3
4 #define MATHLIB_VERSION "1.0.0"
5
6 double add(double a, double b);
7 double subtract(double a, double b);
8
9 #endif
10
```

A screenshot of a code editor window titled 'mathlib.c'. The editor has a dark theme and a toolbar at the top with 'Open', 'Save', and window control buttons. The code is as follows:

```
1
2 #include "mathlib.h"
3
4 double add(double a, double b) {
5     return a + b;
6 }
7
8 double subtract(double a, double b) {
9     return a - b;
10 }
11
```

b.) Create multiple versions of the dynamic library with different implementations

```
mathlib_v2.h
1 #ifndef MATHLIB_V2_H
2 #define MATHLIB_V2_H
3
4 #define MATHLIB_VERSION "2.0.0"
5
6 double add(double a, double b);
7 double subtract(double a, double b);
8 double multiply(double a, double b);
9
10 #endif
11
```

```
mathlib_v2.c
1 #include "mathlib_v2.h"
2
3 double add(double a, double b) {
4     return a + b + 1;
5 }
6 double subtract(double a, double b) {
7     return a - b - 1;
8 }
9
10 double multiply(double a, double b) {
11     return a * b;
12 }
13
```

c.) Write a program that can work with different versions of the library

```
Open main.c Save
1 #include <stdio.h>
2 #include <dlfcn.h>
3
4 int main() {
5     void *handle;
6     char *error;
7     handle = dlopen("./libmathlib_v2.so", RTLD_LAZY);
8     if (!handle) {
9         fprintf(stderr, "Error loading library: %s\n", dlerror());
10        return 1;
11    }
12    double (*add)(double, double) = dlsym(handle, "add");
13    double (*subtract)(double, double) = dlsym(handle, "subtract");
14    const char *version = MATHLIB_VERSION;
15
16    if ((error = dlerror()) != NULL) {
17        fprintf(stderr, "Error finding symbols: %s\n", error);
18        dlclose(handle);
19        return 1;
20    }
21    printf("Library Version: %s\n", version);
22    printf("Add: %.2f\n", add(5.0, 3.0));
23    printf("Subtract: %.2f\n", subtract(5.0, 3.0));
24    dlclose(handle);
25    return 0;
26 }
27
```

## 2. Performance Analysis:

a. Create a benchmark program that compares the performance of static vs. dynamic linking

```
Open benchmark.c Save
1 #include <stdio.h>
2 #include <time.h>
3 #include "mathlib.h"
4
5 void benchmark() {
6     clock_t start = clock();
7
8     for (int i = 0; i < 1000000; i++) {
9         add(i, i + 1);
10        subtract(i, i + 1);
11    }
12
13    clock_t end = clock();
14    printf("Execution Time: %.2f seconds\n", (double)(end - start) / CLOCKS_PER_SEC);
15 }
16
17 int main() {
18     benchmark();
19     return 0;
20 }
21
```

```
(kali㉿kali)-[~]
$ gcc -c mathlib.c -o mathlib.o
ar rcs libmathlib.a mathlib.o

(kali㉿kali)-[~]
$ gcc -c -fPIC mathlib.c -o mathlib.o
gcc -shared -o libmathlib.so mathlib.o

(kali㉿kali)-[~]
$
```

```
^L: Command not found

(kali㉿kali)-[~]
$ gcc -o benchmark_static benchmark.c -L. -lmathlib -static

(kali㉿kali)-[~]
$ gcc -o benchmark_dynamic benchmark.c -L. -lmathlib
make-lesso...

(kali㉿kali)-[~]
$
```

b. Measure and analyze:

i. Load time differences

```
(kali㉿kali)-[~]
$ time ./program
zsh: no such file or directory: ./program

real    0.00s
user    0.00s
sys     0.00s
cpu     73%
```

ii. Memory usage



```
(kali@kali)-[~]
$ /usr/bin/time -v ./program

/usr/bin/time: cannot run ./program: No such file or directory
Command exited with non-zero status 127
Command being timed: "./program"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 70%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1252
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 35
Voluntary context switches: 1
Involuntary context switches: 0
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 127
```

### 3. Complex Library Dependencies:

#### a. Create a library that depends on another library

```
liba.h
1 #ifndef LIBA_H
2 #define LIBA_H
3
4 int add(int a, int b);
5 int subtract(int a, int b);
6
7 #endif
8
```

```
liba.c
1 #include "liba.h"
2
3 int add(int a, int b) {
4     return a + b;
5 }
6
7 int subtract(int a, int b) {
8     return a - b;
9 }
10
```

```
(kali@kali)-[~]
$ gcc -c liba.c -o libA.o
gcc -shared -o libA.so libA.o

(kali@kali)-[~]
$
```

```
libb.h
1 #ifndef LIBB_H
2 #define LIBB_H
3
4 int multiply(int a, int b);
5
6 #endif
7
```

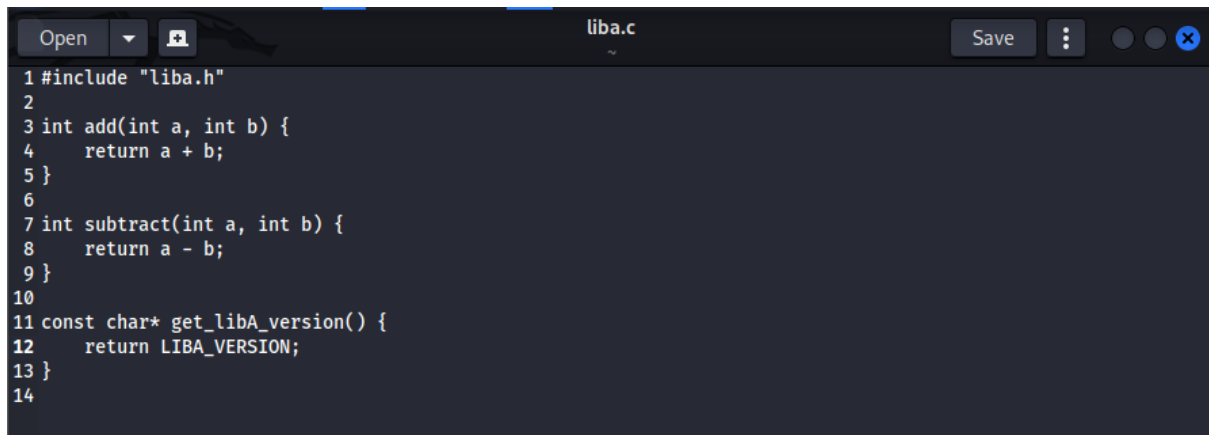
```
libb.c
1 #include "liba.h"
2 #include "libb.h"
3
4 int multiply(int a, int b) {
5     int result = 0;
6     for (int i = 0; i < b; i++) {
7         result = add(result, a);
8     }
9     return result;
10 }
11
```

```
(kali@kali)-[~]
$ gcc -c libb.c -o libB.o
gcc -shared -o libB.so libB.o -L. -la

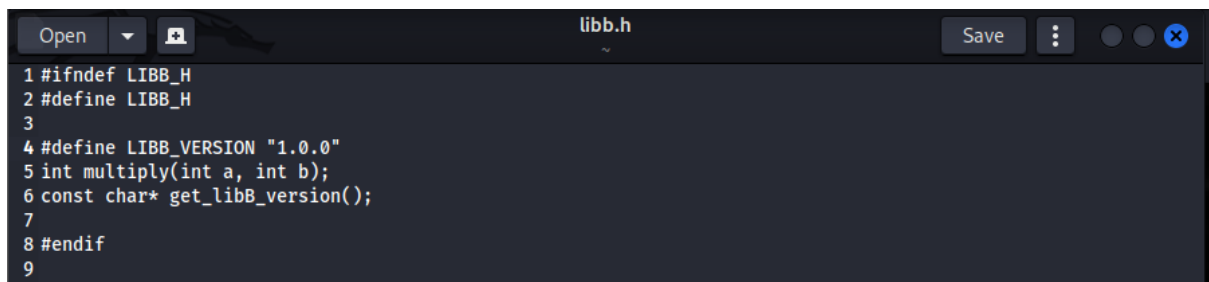
(kali@kali)-[~]
$
```

b. Implement proper version checking and dependency management

```
liba.h
1 #ifndef LIBA_H
2 #define LIBA_H
3
4 #define LIBA_VERSION "1.0.0"
5 int add(int a, int b);
6 int subtract(int a, int b);
7 const char* get_libA_version();
8
9 #endif
10
```



```
1 #include "liba.h"
2
3 int add(int a, int b) {
4     return a + b;
5 }
6
7 int subtract(int a, int b) {
8     return a - b;
9 }
10
11 const char* get_libA_version() {
12     return LIBA_VERSION;
13 }
14
```



```
1 #ifndef LIBB_H
2 #define LIBB_H
3
4 #define LIBB_VERSION "1.0.0"
5 int multiply(int a, int b);
6 const char* get_libB_version();
7
8 #endif
9
```

c. Handle circular dependencies and resolve them