Libraries are collections of precompiled code that can be used by programs to avoid having to rewrite common functions. There are two types of libraries: static libraries and dynamic libraries.

Static Libraries: These are archives of object files. When you link your program with a static library, the code from the library is copied into your executable file. This makes the program larger, but it no longer depends on the library being present after the program is compiled.

Dynamic Libraries (Shared Libraries): These are separate files that are loaded at runtime. The executable doesn't contain the library code itself but refers to the dynamic library, which is linked at runtime.

----------------------------------------------------------------------------------------------------------------------------------------------

Step 1: Create a Source File for the Library

Create a file called mylib.c that contains some basic functions.

// mylib.c

int add(int a, int b) {

    return a + b;

}

int subtract(int a, int b) {

    return a - b;

}

Step 2: Compile the Library to an Object File

You will compile the mylib.c source file to an object file (mylib.o) using the following command:

gcc -c mylib.c -o mylib.o

Step 3: Create the Static Library

After you have the object file, you can create the static library using the ar command:

ar rcs libmylib.a mylib.o

The command ar rcs libmylib.a mylib.o creates or updates a static library (libmylib.a) by adding the object file mylib.o to it and ensures that the library has an index to speed up linking later. Static libraries are used in the linking process to combine reusable code into a program.

r:replace

c:create

s:index

Step 4: Create the Main Program

Now, create the main program that will use the static library. Let's call it main.c.

// main.c

#include<stdio.h>

extern int add(int, int);

extern int subtract(int, int);

int main() {

    int result\_add = add(3, 5);

    printf("Sum=%d\n",result\_add);

    int result\_subtract = subtract(10, 4);

    printf("Sub=%d\n",result\_subtract);

    return 0;

}

Note that extern is used to tell the compiler that the functions are defined elsewhere (in the static library).

Step 5: Compile the Main Program with the Static Library

You need to compile main.c and link it with the static library:

gcc main.c -L. -lmylib -o main

The -L. option tells the compiler to look for libraries in the current directory, and -lmylib links the program with the static library libmylib.a.

Step 6: Run the Program

Now, you can run your program:

./main

-------------------------------------------------------------------------------------------------------------------------------------------

Dynamic Library

Step 1: Create a Source File for the Library

We will reuse the same mylib.c file for the dynamic library.

// mylib.c

int add(int a, int b) {

    return a + b;

}

int subtract(int a, int b) {

    return a - b;

}

Step 2: Compile the Dynamic Library

To compile the dynamic library, use the following command to create a shared object file (libmylib.so):

gcc -fPIC -shared mylib.c -o libmylib.so

The -fPIC option tells the compiler to generate position-independent code, and -shared tells the compiler to create a shared library.

Step 3: Create the Main Program

Create the main program as before (main.c):

// main.c

#include<stdio.h>

extern int add(int, int);

extern int subtract(int, int);

int main() {

    int result\_add = add(3, 5);

    printf("Sum=%d\n",result\_add);

    int result\_subtract = subtract(10, 4);

    printf("Sub=%d\n",result\_subtract);

    return 0;

}

Step 4: Compile the Main Program with the Dynamic Library

To compile the main program and link it with the dynamic library, use:

gcc main.c -L. -lmylib -o main

Step 5: Set the Library Path and Run the Program

To run the program, you need to make sure the dynamic library is found. You can set the LD\_LIBRARY\_PATH environment variable to the directory where libmylib.so is located:

export LD\_LIBRARY\_PATH=.:$LD\_LIBRARY\_PATH

Now, you can run the program:

./main

The program will use the dynamic library at runtime.

Key Differences

Static Library: The code is copied into the executable, making it larger but self-contained.

Dynamic Library: The code is loaded at runtime, keeping the executable smaller and allowing for easy updates to the library without

recompiling the program.

-------------------------------------------------------------------------------------------------------------------------------------------

Linker and Memory Layouts

demo\_program.c

#include <stdio.h>

#include <stdlib.h>

int global\_var = 10;  // Data Section

int uninitialized\_global\_var;  // BSS Section

void demo\_function() {

    static int static\_var = 5;  // Data Section

    int local\_var = 20;  // Stack Section

    printf("Inside demo\_function:\n");

    printf("Local variable (stack): %d\n", local\_var);

 printf("Static variable (data): %d\n", static\_var);

    printf("Global variable (data): %d\n", global\_var);

    printf("Uninitialized global variable (bss): %d\n", uninitialized\_global\_var);

}

int main() {

    int \*heap\_var = malloc(sizeof(int));  // Heap Section

    \*heap\_var = 30;

    printf("Inside main:\n");

    printf("Heap variable: %d\n", \*heap\_var);

    printf("Global variable (data): %d\n", global\_var);

    printf("Uninitialized global variable (bss): %d\n", uninitialized\_global\_var);

    demo\_function();

    free(heap\_var);  // Freeing the dynamically allocated memory

    return 0;

}

gcc -o demo\_program demo\_program.c

nm demo\_program

-------------------------------------------------------------------------------------------------------------------------------------------