MBARARA UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF COMPUTING AND INFORMATICS

COURSE UNIT: SOFTWARE ENGINEERING INDUSTRIAL MINI PROJECT II

COURSE CODE: SWE4106

Academic Year: 2024/2025

Semester: One

Student name:	**BWESIGYE TREASURE**
Regno:	**2021/BSE/145/PS**
Student number:	**2100603048**

Readme/Installation Guide

Installation Steps

Building the Library

1. Clone the repository:

   https://github.com/treasure16522/Portable-library.git

   cd Portable-library

2. Compile the library:
   - For Linux/macOS:

     gcc -shared -o libmatrix.so -fPIC mylibrary.c

   - For Windows:

     gcc -shared -o matrix.dll mylibrary.c

Using in Different Languages

Python

1. Install **ctypes** if not already available.
2. Use this example:
3. **import ctypes**
4.
5. **lib = ctypes. CDLL('./libmatrix.so')**
6. **lib.mat_mult.restype = None**
   # Example usage

Rust

1. Add to **Cargo.toml:**
2. **[dependencies]**
   libc = "0.2"

3. Use **libc** to call functions.

C++

1. Include the header file:
2. **extern "C" {**
3. **void mat_mult(double* A, double* B, double* C, int n);**
   }

4. Link with the shared library during compilation.

Steps to Use the Library in Java

1. Create the Native Library

- Compile your C library into a shared object or dynamic link library:

  On Linux/macOS:

  bash
  Copy code

  ```
  gcc -shared -o libmatrix.so -fPIC mylibrary.c
  ```

  On Windows:

  cmd
  Copy code

  ```
  gcc -shared -o matrix.dll mylibrary.c
  ```

2. Write a Java Wrapper Class

- In Java, you create a wrapper class that uses **System.loadLibrary()** to load your shared library at runtime.
- You declare the native methods in the Java class using the **native** keyword.

3. Generate JNI Headers

- Use the **javac** compiler to compile your Java wrapper class and generate a .class file.
- Use the **javah** tool (or its equivalent in modern JDKs, like **javac -h**) to generate a JNI header file. This file defines the interface for Java to call your native methods.

Example Command:

bash
Copy code

```
javac -h . WrapperClass.java
```

4. Implement the JNI Functions

- Implement the JNI functions in C. These functions will bridge the calls between Java and your existing library functions.
- For example:
  - **mat_mult** in Java will map to **Java_PackageName_WrapperClass_matMult** in C.

5. Compile and Link the JNI Implementation

- Compile the JNI implementation along with your library to ensure seamless integration.

6. Run the Java Program

- Set the **java.library.path** system property to include the directory where your shared library is located:

  On Linux/macOS:

  bash
  Copy code

  ```
  java -Djava.library.path=. YourJavaProgram
  ```

  On Windows:

  cmd
  Copy code

  ```
  java -Djava.library.path=. YourJavaProgram
  ```

To run the provided test cases in **C programming language** (**fourier.c** and **matrix.c**) using my library (**mylibrary.dll** on Windows or **mylibrary.so** on Linux/macOS), you can follow these steps:

1. Share the Compiled Library Only

- Provide the shared object file (**mylibrary.so** for Linux/macOS or **mylibrary.dll** for Windows) without sharing the source code (**mylibrary.c**).
- Share the **mylibrary.h** header file, as it defines the function prototypes required for test cases to use the library.

2. Setup Environment

Linux/macOS:

1. Place the **mylibrary.so** file in a known directory, e.g., **/usr/local/lib** or the current directory.
2. Ensure the directory containing **mylibrary.so** is in the library path:
3. **export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.**

Windows:

1. Place the **mylibrary.dll** file in the same directory as the test executable or in a directory listed in the system's PATH variable.

3. Compile Test Cases

- Use the **gcc** compiler to compile the test cases (**fourier.c** and **matrix.c**) into executables. Link against the shared library without needing the library's source code.

Linux/macOS:

```
gcc -o fourier_test fourier.c -L. -lmylibrary -lm
gcc -o matrix_test matrix.c -L. -lmylibrary -lm
```

Windows:

```
gcc -o fourier_test.exe fourier.c mylibrary.dll
gcc -o matrix_test.exe matrix.c mylibrary.dll
```

4. Run the Test Cases

After compiling, run the test cases, ensuring the shared library is accessible.

Linux/macOS:

```
./fourier_test
./matrix_test
```

Windows:

```
fourier_test.exe
matrix_test.exe
```

5. Verify Results

- The outputs of the test cases (e.g., matrices for **matrix.c,** transformed data for **fourier.c**) will verify the functionality of **mylibrary.dll** or **mylibrary.so .**
- Since the test executables link to the compiled library, the library's internal source code (**mylibrary.c**) remains hidden from the user.

Key Points:

- **Binary Distribution**: By providing only the compiled **.so** or **.dll** files, you retain ownership of your source code.
- **Interface Sharing**: Only share the **mylibrary.h** file to allow test cases to interact with the library.
- **Cross-Platform Use**: Provide both **.so** and **.dll** versions for compatibility across Linux/macOS and Windows.