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**

Installation Guide

Using C with Notepad++ and a C Compiler on Windows

This guide outlines two approaches to set up your Windows environment for C development:

1. Lightweight Setup with Notepad++ and MinGW

- **Notepad++ Installation:**
    - Download Notepad++ from the official website: https://notepad-plus-plus.org/downloads/
    - Run the installer. (Notepad++ is a user-friendly text editor, but other options like Code::Blocks or IntelliJ IDEA exist.)
- **MinGW Installation:**
    - Download the MinGW installer from https://osdn.net/projects/mingw/releases/.
    - During installation, select the essential package "mingw32-base."
    - Crucially, add the MinGW bin directory to your system's PATH environment variable. This allows you to run MinGW tools like the gcc compiler directly from the command prompt.
- **Notepad++ Configuration:**
    - Open Notepad++.
    - Go to Settings -> Preferences -> Language.
    - Select C for syntax highlighting to aid code readability.
- **Writing a C Program:**
    - Create a new file (e.g., world.c) and write your C code.
    - Sample code:

C

```
include <stdio.h>


int main() {
   printf("Hello, World!\n");
   return 0;
}
```

- **Compiling and Running:**
    - Open a command prompt window.

- Navigate to the directory where you saved world.c using the cd command (e.g., cd C:\path\to\your\file).
- Compile the code using the gcc command:

Bash

gcc world.c -o world.exe

This creates an executable file named `world.exe`.

Run the program by typing:

Bash

world.exe

2. Building and Using a Custom C Library

Note: This section assumes you have a basic understanding of C compilation and linking.

- **Getting the Library:**
  - Clone the library repository from https://github.com/treasure16522/Portable-library.git
  - Navigate to the cloned directory using cd Portable-library.
- **Building the Library:**
  - **Linux/macOS:**
    - Use the gcc compiler to create a shared library:

Bash

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

Windows:

Use the `gcc` compiler to create a DLL:

Bash

gcc -shared -o matrix.dll mylibrary.c

Using the Library in Different Languages:

Python:

1. Install ctypes:

Bash

pip install ctypes

2. Example Usage:

Python

```
import ctypes

lib = ctypes.CDLL('./libmatrix.so')   Adjust path if needed
lib.mat_mult.restype = None
```

Example usage with the library functions

Rust:

1. **Add to Cargo.toml:**

Ini, TOML

```
[dependencies]
libc = "0.2"
```

2. **Use libc to call library functions.**

C++:

1. **Include the header file:**

C++

```
extern "C" {
void mat_mult(double A, double B, double C, int n);
}
```

2. **Link with the shared library during compilation.**

Java:

1. **Create the Native Library:**
    - Compile your C library into a shared object or DLL as before (see Linux/Windows instructions).
2. **Write a Java Wrapper Class:**
    - Create a Java class that:
        - Uses System.loadLibrary() to load the shared library at runtime.
        - Declares native methods with the native keyword.
3. **Generate JNI Headers:**
    - Use javac to compile your Java wrapper class and generate a .class file.

- Use javah or javac -h to generate a JNI header file for native method interactions.

4. **Implement the JNI Functions:**
    - Implement the JNI functions in C to bridge calls between Java and your library functions.
5. **Compile and Link the JNI Implementation:**
    - Compile the JNI implementation along with your library for seamless integration.
6. **Run the Java Program.**