ASSIGNMENT B1

TITLE: Dynamic Link Library (DLL).

PROBLEM STATEMENT:

Write a program to create a Dynamic Link Library for any mathematical operation and write an application program to test it. (Java Native Interface / Use VB or VC++)

OBJECTIVE:

- Understand the concept of Dynamic Link Libraries.
- Creating a Dynamic Link Library.

S/W PACKAGES AND HARDWARE REQUIREMENTS:

- 64-bit Windows 10 OS
- Visual Studio IDE
- 64-bit architecture I3 or I5 machines
- Java Native Interface

OUTCOME:

We will be able to

- Create a Dynamic Link Library for a mathematical function.

THEORY:

A *dynamic-link library* (DLL) is a module that contains functions and data that can be used by another module (application or DLL).

A DLL can define two kinds of functions: exported and internal. The exported functions are intended to be called by other modules, as well as from within the DLL where they are defined. Internal functions are typically intended to be called only from within the DLL where they are defined. Although a DLL can export data, its data is generally used only by its functions. However, there is nothing to prevent another module from reading or writing that address.

DLLs provide a way to modularize applications so that their functionality can be updated and reused more easily. DLLs also help reduce memory overhead when several applications use the same functionality at the same time, because although each application receives its own copy of the DLL data, the applications share the DLL code.

DLL's and Memory Management:

Every process that loads the DLL maps it into its virtual address space. After the process loads the DLL into its virtual address, it can call the exported DLL functions.

The system maintains a per-process reference count for each DLL. When a thread loads the DLL, the reference count is incremented by one. When the process terminates, or when the reference count becomes zero (run-time dynamic linking only), the DLL is unloaded from the virtual address space of the process.

Like any other function, an exported DLL function runs in the context of the thread that calls it. Therefore, the following conditions apply:

- The threads of the process that called the DLL can use handles opened by a DLL function. Similarly, handles opened by any thread of the calling process can be used in the DLL function.
- The DLL uses the stack of the calling thread and the virtual address space of the calling process.
- The DLL allocates memory from the virtual address space of the calling process.

Creating a simple DLL:

Step 1: Create the project for the DLL.

1. Let's create a simple Win32 Console DLL: Visual Studio > File > New Project > Win32 Console Application
2. Name it something sexy like MyDLL and press OK.
3. Press Next in the following screen and choose DLL. Leave other options untouched.
4. Press Finish.

Step 2: Add the code for the DLL.
I will create a new class and add some simple code.

1. Right-click on Project name > Add > Class > Name it something like HelloDLL

2. Add the code that follows.
3. To create the DLL export library: Project > MyDLL properties… > Configuration Properties > C/C++ >Preprocessor > Append, or insert, "DLLDIR_EX" (without the quotation marks) to the Preprocessor Definition text box > OK. This will prevent compiler assumptions and warnings. [1]
4. Build > Build Solution
5. Your DLL should be ready. You should have files MyDLL.dll and MyDLL.lib in directory project-directory\Debug.

HelloDLL.h

```
#pragma once

//more about this in reference 1
#ifdef DLLDIR_EX
  #define DLLDIR  __declspec(dllexport)   // export DLL information

#else
  #define DLLDIR  __declspec(dllimport)   // import DLL information

#endif

class DLLDIR HelloDLL
{
public:
   HelloDLL(void);
   ~HelloDLL(void);

     void hello();
    static void helloStatic();

};
```

HelloDLL.cpp

```
#include "StdAfx.h"
#include "HelloDLL.h"
#include <iostream>

using namespace std;

HelloDLL::HelloDLL(void)
{
}

HelloDLL::~HelloDLL(void)
{
}

void HelloDLL::hello()
{
   cout << "Hello World of DLL" << endl;
}

void HelloDLL::helloStatic()
{
   cout << "Hello World of DLL static" << endl;
}
```

Step 3: Use the DLL in a new project.

1. Create a new project: Visual Studio > File > New Project > Win32 Console Application. Name it Use of MyDLL and press OK.
2. Press Next.
3. Choose Console application.
4. Press Finish.

Step 4: Logistics for the new project.

1. Add file MyDLL.lib in your project: Project > Properties > Configuration Properties > Linker > Input > Write the full path (including the filename) in quotes. Example: "C:\Examples\MyDLL\Debug\MyDLL.lib"
2. Move file MyDLL.dll in the same directory as the source code of your project Use of MyDLL.
3. Add header file: Project > Properties > Configuration Properties > C/C++ > Additional Include Directories > Add the directory where the file HelloDLL.h is located.

Step 5: Test your DLL.
I used the following code:

```
#include "stdafx.h"
#include "HelloDLL.h"

int _tmain(int argc, _TCHAR* argv[])
{

    HelloDLL helloDLL;
    helloDLL.hello();
    HelloDLL::helloStatic();

   getchar();

   return 0;
}
```

CONCLUSION:

Thus, we successfully implemented Dynamic Link Library for a mathematical operation.