We have 3 files:

1. **The file that contains code we want to load dynamically**
2. **The entry point of the file**
3. **The build.bat file**

In file 1:

Header:

- **Make a macro that defines a function with the name we pass it:**
  `#define NAME_OF_FUNCTION(name) void name(*args*)`
- **Create a typedef that says: there is a function of type void, that takes these *args* and is called my_function_name:**
  `typedef NAME_OF_FUNCTION(my_function_name);`
  This basically expands to:
  `typedef void my_name(*args*)`
  So when we write:
  `my_function_name *NameOfFunction;`
  We declare a function pointer that points to a function of type void and takes the *args* we have put in the define in step 1.
- **Declare a stub function:**
  `NAME_OF_FUNCTION(NameOfFunctionStub) {}`

CPP:

- **Define the actual function the we are going to call dynamically:**

```
extern "C" NAME_OF_FUNCTION(NameOfFunction)
{
      …
}
```

In file 2:

Header:

- **We make a struct that contains info about the dynamically loaded code:**

```
struct my_code
{
    HMODULE dll;
    FILETIME writeTime;
    my_function_name *NameOfFunction;
    bool isValid;
};
```

CPP:

- **We add the following functions:**

```
static FILETIME Win32GetWriteTime(char *filepath)
{
    FILETIME result = {};
    WIN32_FIND_DATA data;
    HANDLE handle = FindFirstFileA(filepath, &data);
    if (handle != INVALID_HANDLE_VALUE)
    {
       result = data.ftLastWriteTime;
       FindClose(handle);
    }
    return result;
}
```

```
static void Win32LoadCode(my_code *code, char *filepath, char *filepathCopy)
{
    Sleep(800); // Wait for dll compile
    code->writeTime = Win32GetWriteTime(filepath);
    CopyFileA(filepath, filepathCopy, FALSE);
    code->dll = LoadLibraryA(filepathCopy);
    if (code->dll)
    {
       code->NameOfFunction = (my_function_name) *)GetProcAddress(code->dll,
"NameOfFunction");
       code->isValid = code->UpdateAndRender;
    }
    if (!code->isValid)
    {
       code->NameOfFunction = NameOfFunctionStub;
    }
}

static void Win32UnloadCode(my_code *code)
{
    if (code->dll)
    {
       FreeLibrary(code->dll);
       code->dll = 0;
    }
    code->isValid = false;
    code->NameOfFunction = NameOfFunctionStub;
}
```

In file 3:

```
cl  %COMPILER_FLAGS% file1.cpp -Fmfile1.map -LD -link -incremental:no -opt:ref
-PDB:file1_%RANDOM%.pdb -EXPORT:NameOfFunction

cl  %COMPILER_FLAGS% file2.cpp -Fmfile2.map -Fe%EXE_NAME% -link %LINKER_FLAGS%
```

WE DONE!

In a similar way, if I need to use a function from a dynamically linked library you follow a similar approach:

```
#define NAME_OF_FUNCTION(name) function_return_type name(*args*)
typedef NAME_OF_FUNCTION(name_of_function);
NAME_OF_FUNCTION(nameOfFunctionStub) { return 0; }
static name_of_function *nameOfFunction_ = nameOfFunctionStub;
#define nameOfFunction nameOfFunction_
```

So basically now we have the definition of the function we want to use and it defaults to a stub function that does nothing, until we load the actual library and “bind” it to the actual address of the function in the dll.

```
static void Win32LoadLibrary()
{
	HMODULE dll = LoadLibraryA(“library.dll”);
	If (dll)
	{
		nameOfFunction = (name_of_function *)GetProcAdress(dll, “nameOfFunction”);
	}
}
```