# Lab Task: Build Automation (Pre-Processors)

## Task 1: File Inclusion

For this task, consider the following code:

```
#include <iostream>
#include <boost/tokenizer.hpp>
#include "my_header_file.hpp"

int main() {
    std::cout << "File Inclusions Test" << std::endl;
    return 0;
}
```

The contents of my_header_file.hpp are:

```
void myFunction();
```

Answer the following questions:

1. What happens if you rename the file to my_header_file.txt and include this file only?
2. What happens if you change “” to < > just like iostream is enclosed in <>.
3. What happens if you do #include “my_header_file.hpp” twice in your code?

Modify the my_header_file.hpp to the following:

```
class MyClass {
public:
    void myFunction();
};
```

And inside your main() function, include the following:

```
MyClass obj;
```

Now, answer the following questions:

4. What happens if you do #include “my_header_file.hpp” twice in your code?

Now, modify the my_header_file.hpp again to the following:

```
#ifndef MY_HEADER_FILE_HPP
#define MY_HEADER_FILE_HPP

class MyClass {
public:
    void myFunction();
};

#endif
```

Now, answer the following questions:

5. What happens if you do #include “my_header_file.hpp” twice in your code?
6. What happens if you rename #ifndef to #ifdef?

## Task 2: Macros and Compile Time Conditional Execution

Modify the two files to the following:

```
#ifndef MY_HEADER_FILE_HPP
#define MY_HEADER_FILE_HPP


#include <iostream>

class MyClass {
public:
 void myFunction(int x) {
    if (x % 2 == 0) {
      std::cout << “even” << std::endl;
    } else {
      std::cout << “odd” << std::endl;
    }
};

#endif
```

And the main file as:

```
#include <iostream>
#include <boost/tokenizer.hpp>
#include "my_header_file.hpp"

int main() {
  Myclass obj;
  obj.myFunction(6);
  return 0;
}
```

As can be seen from the code, it is just printing odd and even status of a number. We need to convert all run time execution to compile time. For this, carry out the following:

1. Create a macro in the header file:

```
#define isodd(param1) param1 % 2
```

2. And modify the if else conditions to be included in respective #if and #endif blocks. Example is below:

```
#if isodd(x) == 0

// code here

#endif
```

3. Is the code giving expected behavior? Think and comment about what the problem is.
4. In the main file, modify obj.myFunction(5); to a compile time variable as:

```
#define x 5

obj.myFunction(5); → Change to → obj.myFunction();
```

5. Is the code giving expected behavior? Think and comment about what the problem is.
6. Now move the `#define x 5` to the my_headerfile.hpp. Is the code giving expected behavior? Comment about the matter.
7. Now delete the line `#define x 5` even from my_header_file.hpp. And instead, run this command as:

```
g++ main.cpp -Dx=5
```

8. Is the code giving expected behavior?

## Task 3: Converting to Library

Modify your header file such that the my_header_file.hpp becomes:

```
#ifndef MY_HEADER_FILE_HPP
#define MY_HEADER_FILE_HPP

#include <iostream>

class MyClass {
public:
 void myFunction(int x);
};

#endif
```

And create a new (3rd) file called my_header_lib.cpp containing:

```
void MyClass::myFunction(int x) {
 if (x % 2 == 0) {
   std::cout << "even" << std::endl;
 } else {
   std::cout << "odd" << std::endl;
 }
}
```

Compile the code as:

```
g++ main.cpp my_header_lib.cpp
```

Make sure the code works. Remove/fix errors yourself.
You are all set to convert the code to libraries now. In the first step, we need to create static libraries.

## Static Library

Compile each of the source codes separately. The output of the 3rd file should be my_header_lib.o. This can be done by passing the -c argument to your compilation.

Then, use the ar command to convert the code into a static library.

```
ar -cvr libMyHeaderLib.a my_header_lib.o
```

When done, compile (1) your main file, (2) your my_header_file.hpp file, (3) Your .a library file into a single package and run. Answer the following questions:

1. What is the directory structure of your code before and after the library creation.
2. What are the CVR arguments?
3. Run the commands below and report their output:
    1. readelf -S my_header_lib.o
    2. readelf -S your_final_file

## Dynamic Library

When compiling the 3rd file, pass the command:

```
gcc -shared my_header_lib.cpp -o libMyHeaderLib.so
```

Use symbolic linking of your code, whichever applicable.

```
gcc -shared -Wl,-soname,libMyHeaderLib.so.1 -o libMyHeaderLib.so.1.0
my_header_lib.cpp

ln -sf libMyHéqderLib.so.1.0 libMyHeaderLib.so.1

ln -sf libMyHeaderLib.so.1.0 libMyHeaderLib.so
```

Compile your main file as:

```
gcc main.cpp -lMyHeaderLib
```

If the above does not work, use LD_LIBRARY_PATH to specify your current directory, or specify the location directl as:

```
gcc main.cpp -L . -lMyHeaderLib
```