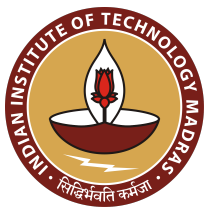
Dealing With Multiple Source Files

Saurav Samantaray

Department of Mathematics

Indian Institute of Technology Madras

September 23, 2024



Dealing with Multiple Source Files

- So far, we have been dealing with a single source file and make into an executable file like:

```
g++ -o HelloWorld.exe HelloWorld.cpp
```

- What really happens in this process is that the C++ file is first compiled to another file called HelloWorld.o, and known as an object file, which is a machine-readable file
- In a second step, the object file is compiled into the executable file and the intermediate object file is deleted
- the executable file HelloWorld is excuted using

```
./HelloWorld.exe
```

- up until this point, we have used a one line compilation command, allowing us to completely ignore the existence of object files
- when compiling multiple files we do, however, need to be aware of the existence of these files

Dealing with Multiple Source Files

- Before we can compile the file main.cpp we first need to compile the array class to create an object file `arrover3.o` associated with this class
- this is done, as above, by using the -c option when compiling:

```
g++ -O -c arrover3.cpp
```

- this produces an object file `arrover3.o`
- we can now compile main.cpp into an object file and then link the two object files to make an executable
- the two compilation commands are now:

```
g++ -O -c main.cpp
g++ -lm -O -o main.exe main.o arrover3.o
```

the code may be run as before by typing

```
./main.exe
```

Using Makefiles

- Suppose we have a code that uses several classes stored in several files
- we would rather not compile all of these classes separately every time one file is modified slightly
- this may be avoided by the use of a Makefile–using this approach only the necessary compilation is carried out
- the following is a Makefile for code UseClasses.cpp that uses two classes, Class1 and Class2

```
Class1.o : Class1.cpp  Class1.hpp
	g++ –c –O  Class1.cpp
Class2.o : Class2.cpp  Class2.hpp
	g++ –c –O  Class2.cpp
UseClasses.o : UseClasses.cpp  Class1.hpp  Class2.hpp
	g++ –c –O  UseClasses.cpp
UseClasses : Class1.o  Class2.o  UseClasses.o
	g++ –O –o  UseClasses  Class1.o  Class2.o  UseClasses.o
```

Using Makefiles

- If the file above is saved as Makefile, then to generate an up-to-date executable file UseClasses we simply type

```
make UseClasses
```

at the command line

- Using this approach only the necessary compilation will be carried out
- line 7 of this Makefile tells the compiler that the executable file UseClasses requires three files: Class1.o, Class2.o and UseClasses.o line 8 gives the rule for compiling the executable file from its dependencies
- line 1 tells the compiler that the file Class1.o depends on the two files Class1.cpp and Class1.hpp
- only if one or both of these files have been changed since the last time this class has been compiled will this class be recompiled using the rule given on line 2
- similar remarks hold for the class Class2
- in line 7, the recompilation of UseClasses.o depends not only on the relevant C++ file, but also on the classes, header files–so that a change in either class interface will result in a recompilation of the file which uses its functionality