**Why to make a makefile:**

When your code has several files(.cpp and .h), you have to type the compiling command every time you change a part of one of your files. makefile is a much better way to compile files.

**How to make a makefile:**

First create a file in the same directory as your program (.cpp and .h) files are located and name it makefile. (Naming it makefile is recommended but any other name can be used.) By typing 'make' the makefile will be executed.

make -f myname

-f means use myname file as a makefile

A makefile is composed of:

target: dependencies

[tab] system command

The target depends on the dependencies. Let’s describe this by using a simple example. Consider our program consisting of these files:

main.cpp myfunction.cpp myfunction.h openfile.cpp openfile.h

our make file could be:

# This is a comment

# targetname is the name of the target (the file we create). Don't forget tab before system command

targetname:

g++ main.cpp myfunction.cpp openfile.cpp -o test

so by typing 'make' all of the files would be compile. However, if we modify one file we have to compile all the files again. There a more efficient way to do this

targetname(could be the exe file name): object files

g++ object files -o exe file

object file1: files with .cpp and .h that the object file depends on

g++ -c files with .cpp

.....

for example:

test: main.o myfunction.o openfile.o

g++ main.o myfunction.o openfile.o -o test

# here, test is the executable file and it depends on the object files main.o myfunction.o openfile.o. At run #time Make compares the time when test was changed with the last time the object file was changed.

#If any object file is newer than the executable file, the executable file will be rebuilt. If the object

# files are not made, Make tries to make them. For example if main.o has not been made, the next

# line will be executed.

main.o: main.cpp myfunction.h openfile.h

g++ -c main.cpp

# main.o depends on main.cpp, myfunctions.h and openfile.h, and in order to make main.o the system # command will be executed. -c tells the compiler that the file is not a complete program, and it just # generates the object code resulting from compiling that file.

myfunction.o: myfunction.cpp myfunction.h

g++ -c myfunction.cpp

# myfunction.o depends on myfunction.cpp and myfunction.h

openfile.o: openfile.cpp openfile.h

g++ -c openfile.cpp

clean:

rm test \*.o

# clean means to delete the executable and object files.

So now if we want to compile the whole thing we should just type 'make' (or 'make -f mymakefile' if the name of your makefile is something other than 'makefile'), if just changes have been made only to myfunction.cpp then we can just compile myfunction, by

make -f myfunction.o

To delete the object and executable files we can type

make clean