make and makefile

If you want to run or update a task when certain files are updated, the **make** utility can come in handy.
The make utility requires a file, **Makefile** (or **makefile**), which defines set of tasks to be executed.

To compile a C++ program, we can do the compilation on the command line like this:

```
c++ convert.cpp -o convert
```

Problem with this approach:

- typing the compilation command each time after the file is updated can be tedious, especially when there are more files involved in a project
- when only one of few of the files in a project is modified, we just want to re-compile those files, not all the files.

Makefile can help solving these problems.

Example one:

```
convert: convert.o                  ← this line defines the dependencies for convert
    c++ convert.o -o convert        <<= a tab key is required at the start of this line

convert.o: convert.cpp              ← this line defines the dependencies for convert.o
    c++ -c convert.cpp              <<= a tab key is required at the start of this line

clean:
    rm *.o                          <<= a tab key is required at the start of this line
```

Example two: a slightly modified version

```
CC=c++
CFLAGS=-I.

convert: convert.o
    $(CC) $(CFLAGS) convert.o -o convert

convert.o: convert.cpp
    $(CC) $(CFLAGS) -c convert.cpp

clean:
    rm *.o
```

How to compile the program using make command and the makefile? Just type "make" at the prompt.

```
$ make
```

This will create the first target in the file, e.g., convert, by running the command c++ convert.o -o convert. Due to defined dependency, this step will in turn create the second target convert.o by running the command c++ -c convert.cpp. The compile option "-c" is to compile the program and generate the object code convert.o.

To remove all the extra files, i.e., the object files *.o, just type the following command: $ make clean

Suppose we are working with the same program that are written in two source files and one header file. In this case, the main function is in one file, named "convertMain.cpp", the use defined functions are in a second file, named "convertFunc.cpp", and the declarations of these functions are in the third file, named "convertHeader.h". (refer to the course web site for these files. Notice the line #include "convertHeader.h". has been included in both .cpp files.)

To compile this program, we can use the following makefile:

```
CC=c++
CFLAGS=-I.

convert: convertMain.o convertFunc.o
	$(CC) $(CFLAGS) convertMain.o convertFunc.o -o convert

convertMain.o: convertMain.cpp
	$(CC) $(CFLAGS) -c convertMain.cpp

convertFunc.o: convertFunc.cpp
	$(CC) $(CFLAGS) -c convertFunc.cpp

clean:
	rm *.o
```

When we discuss c++ object oriented programming and introduce "class", we will be working with at least 3 files, the client program MainProgram.cpp, the class header file Sphere.h, and the class implementation file Sphere.cpp. This is the makefile we will use to compile the program and generate the executable RunProg:

```
CC=c++
CFLAGS=-g

## Create the executable RunProg
RunProg:Sphere.o MainProgram.o
	$(CC) Sphere.o MainProgram.o -o RunProg

## Create the object file for the ADT Sphere
Sphere.o:Sphere.cpp
	$(CC) $(CFLAGS) -c Sphere.cpp

MainProgram.o:MainProgram.cpp
	$(CC) $(CFLAGS) -c MainProgram.cpp

clean:
	rm *.o
```

Refer to the course page for these files.