Makefile + 2D matrices

A Simple Makefile Tutorial

- **hellomake.c**

```
#include <hellomake.h>
int main() {
 // call a function in another file
myPrintHelloMake();
 return(0); }
```

- **hellofunc.c**

```
#include <stdio.h>
#include <hellomake.h>
 void myPrintHelloMake(void) {
 printf("Hello makefiles!\n"); return; }
```

- **hellomake.h**

```
/* example include file */
void myPrintHelloMake(void);
```

- Normally, you would compile this collection of code by executing the following command:

```
gcc -o hellomake hellomake.c hellofunc.c
```

if you lose the compile command or switch computers you have to retype it from scratch, which is inefficient at best.

if you are only making changes to one .c file, recompiling all of them every time is also time-consuming and inefficient. So, it's time to see what we can do with a makefile.

- Make file

```
hellomake: hellomake.c hellofunc.c
gcc -o hellomake hellomake.c hellofunc.c
```

- f you put this rule into a file called Makefile or makefile and then type make on the command line it will execute the compile command as you have written it in the makefile. Note that make with no arguments executes the first rule in the file. Furthermore, by putting the list of files on which the command depends on the first line after the :, make knows that the rule hellomake needs to be executed if any of those files change. Immediately, you have solved problem #1 and can avoid using the up arrow repeatedly, looking for your last compile command. However, the system is still not being efficient in terms of compiling only the latest changes.
- One very important thing to note is that there is a tab before the gcc command in the makefile. There must be a tab at the beginning of any command, and make will not be happy if it's not there.

A deeper look into makefile

- main.cpp

```
#include
     “message.h”
#include <cstdlib>
using namespace
     std;
int main()
{
message m;
m.printMessage();
return 0;
}
```

- message.h

```
#ifndef MESSAGE_H
#define MESSAGE_H
class message{
void printMessage();
};
#endif //MESSAGE_H
```

- message.cpp

```
#include <iostream>
#include “message.h”
using namespace std;
void message::printMessage()
{
Cout <<“Makefile Example\n”;
}
```

Makefile

- Conventional form from terminal:

```
g++ main.cpp message.cpp
```

The .h file is included in those files so we do not need to mention it above. That line of code will generate an executable a.o .

Makefile

- Basic structure of makefile:

```
target: dependencies (this line should start from left)
	perform action (start this line after one tab)
```

Makefiles are white space sensitive.

Makefile:

```
output: main.o message.o
	g++ main.o message.o –o output
Main.o: main.cpp
	g++ -c main.cpp (-c says do not create executable and only compile to a object
	file ie main.o)
Message.o: message.cpp message.h
	g++ -c message.cpp
Clean:
	rm *.o output
```

Now do these steps from teminal

- Type ls and press enter.
- Type make and press enter.
- Type ls and press enter.
- Type ./output and press enter
- Clear screen now
- Invoke make again
- What happened?
- Run “make clean”
- Invoke ls
- What happened?
- Type make and press enter.
- Type ls and press enter
- Go to message.cpp and change the message.
- Type make and press enter
- How many rules got executed?
- Run ./output
- Run “touch main.cpp”
- Run make
- What happened?

How to populate a 2D matrix

```
#include<stdio.h>
 int main(){
 /* 2D array declaration*/
 int disp[2][3];
/*Counter variables for the loop*/
 int i, j;
for(i=0; i<2; i++) {
     for(j=0;j<3;j++) {
        printf("Enter value for disp[%d][%d]:", i, j);
        scanf("%d", &disp[i][j]);
        }
 }
//Displaying array elements
printf("Two Dimensional array elements:\n");
 for(i=0; i<2; i++) {
         for(j=0;j<3;j++) {
                    printf("%d ", disp[i][j]);
                    if(j==2){
                         printf("\n"); }
        }
 }
return 0;
}
```