

CSCI 1061U

Programming Workshop 2

Makefiles

Today's Lecture

- GNU Make utility
- Managing software builds

Building executables

- The story so far: generating an executable from a single source file

```
#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
    cout << "Hello world\n";
    return 0;
}
```

main.cpp

```
> g++ main.cpp -o helloworld
```

Building executables

- What if our program contains multiple files?

```
#include "greetings.h"  
#include <iostream>  
using namespace std;  
  
int main(int argc, char** argv) {  
    say_greetings();  
    return 0;  
}
```

main.cpp

```
#include <iostream>  
using namespace std;  
  
void say_greetings() { cout << "Hello world.\n"; }
```

greetings.cpp

```
#ifndef _greetings_h_  
#define _greetings_h_  
void say_greetings();  
#endif
```

greetings.h

Building executables

- What if our program contains multiple files?

```
#include "greetings.h"  
#include <iostream>  
using namespace std;  
  
int main(int argc, char** argv) {  
    say_greetings();  
    return 0;  
}
```

main.cpp

```
#include <iostream>  
using namespace std;  
  
void say_greetings() { cout << "Hello world.\n"; }
```

greetings.cpp

```
#ifndef _greetings_h_  
#define _greetings_h_  
void say_greetings();  
#endif
```

greetings.h

```
> g++ main.cpp greetings.cpp -o helloworld
```

Building executables

- What if our program contains multiple files?

```
#include "greetings.h"
#include <iostream>
using namespace std;

int main(int argc, char** argv) {
    say_greetings();
    return 0;
}
```

main.cpp

```
#include <iostream>
using namespace std;

void say_greetings() {
    cout << "Hello World!" << endl;
}
```

```
#ifndef _greetings_h_
#define _greetings_h_
void say_greetings();
#endif
```

greetings.h

This can get quickly out of hand as the number of files grow, e.g., Linux has roughly 15,000 source files.

```
> g++ main.cpp greetings.cpp -o helloworld
```

Makefiles

- *make* utility will enable you automatically build executable from your source code
- You describe how you want your executable to be built using a Makefile, which *make* utility uses to build the executable

The *make* utility

- Reading the default makefile in the current directory (this reads Makefile stored in the current directory)

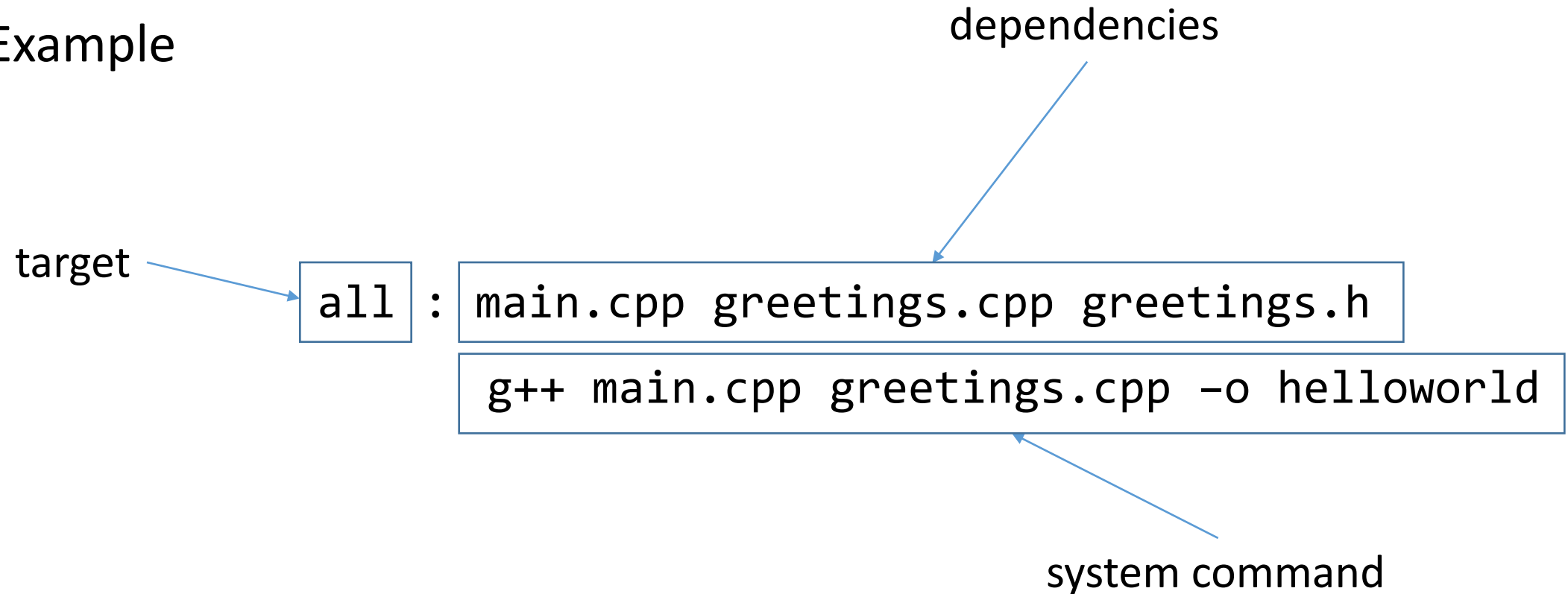
```
> make
```

- Reading a specific makefile

```
> make -f MyMakefile
```



Basic Makefile

- Example



A Basic Makefile

- The following makefile has three targets
 - *all* – the default target, notice that it's dependency is our executable
 - *helloworld* – our executable, it depends upon main.cpp, greetings.cpp and greetings.h files
 - *clean* – a target to that can be used to delete the executable

A screenshot of a text editor window titled 'Makefile' with a status bar indicating 'UNREGISTERED'. The editor shows a Makefile with three targets: 'all', 'helloworld', and 'clean'. The 'all' target depends on 'helloworld'. The 'helloworld' target depends on 'main.cpp', 'greetings.cpp', and 'greetings.h', with the command 'g++ main.cpp greetings.cpp -o helloworld'. The 'clean' target has the command 'rm helloworld'. The status bar at the bottom shows 'Line 1, Column 1', '0 misspelled words', 'Tab Size: 4', and 'Makefile'.

```
1 all : helloworld
2
3 helloworld: main.cpp greetings.cpp greetings.h
4             g++ main.cpp greetings.cpp -o helloworld
5
6 clean:
7             rm helloworld
```

A Basic Makefile

- The following makefile has three targets
 - *all* – the default target, notice that it's dependency is our executable

> make

A screenshot of a text editor window titled 'Makefile' with a status bar indicating 'UNREGISTERED'. The editor shows a Makefile with three targets: 'all', 'helloworld', and 'clean'. The 'all' target depends on 'helloworld'. The 'helloworld' target depends on 'main.cpp', 'greetings.cpp', and 'greetings.h', with the command 'g++ main.cpp greetings.cpp -o helloworld'. The 'clean' target has the command 'rm helloworld'. The status bar at the bottom shows 'Line 1, Column 1', '0 misspelled words', 'Tab Size: 4', and 'Makefile'.

```
1 all : helloworld
2
3 helloworld: main.cpp greetings.cpp greetings.h
4             g++ main.cpp greetings.cpp -o helloworld
5
6 clean:
7         rm helloworld
```

A Basic Makefile

- The following makefile has three targets
 - *helloworld* – our executable, it depends upon main.cpp, greetings.cpp and greetings.h files

```
> make helloworld
```



```
Makefile
UNREGISTERED

1 all : helloworld
2
3 helloworld: main.cpp greetings.cpp greetings.h
4     g++ main.cpp greetings.cpp -o helloworld
5
6 clean:
7     rm helloworld

Line 1, Column 1    0 misspelled words    Tab Size: 4    Makefile
```

A Basic Makefile

- The following makefile has three targets
 - clean – a target to that can be used to delete the executable

```
> make clean
```

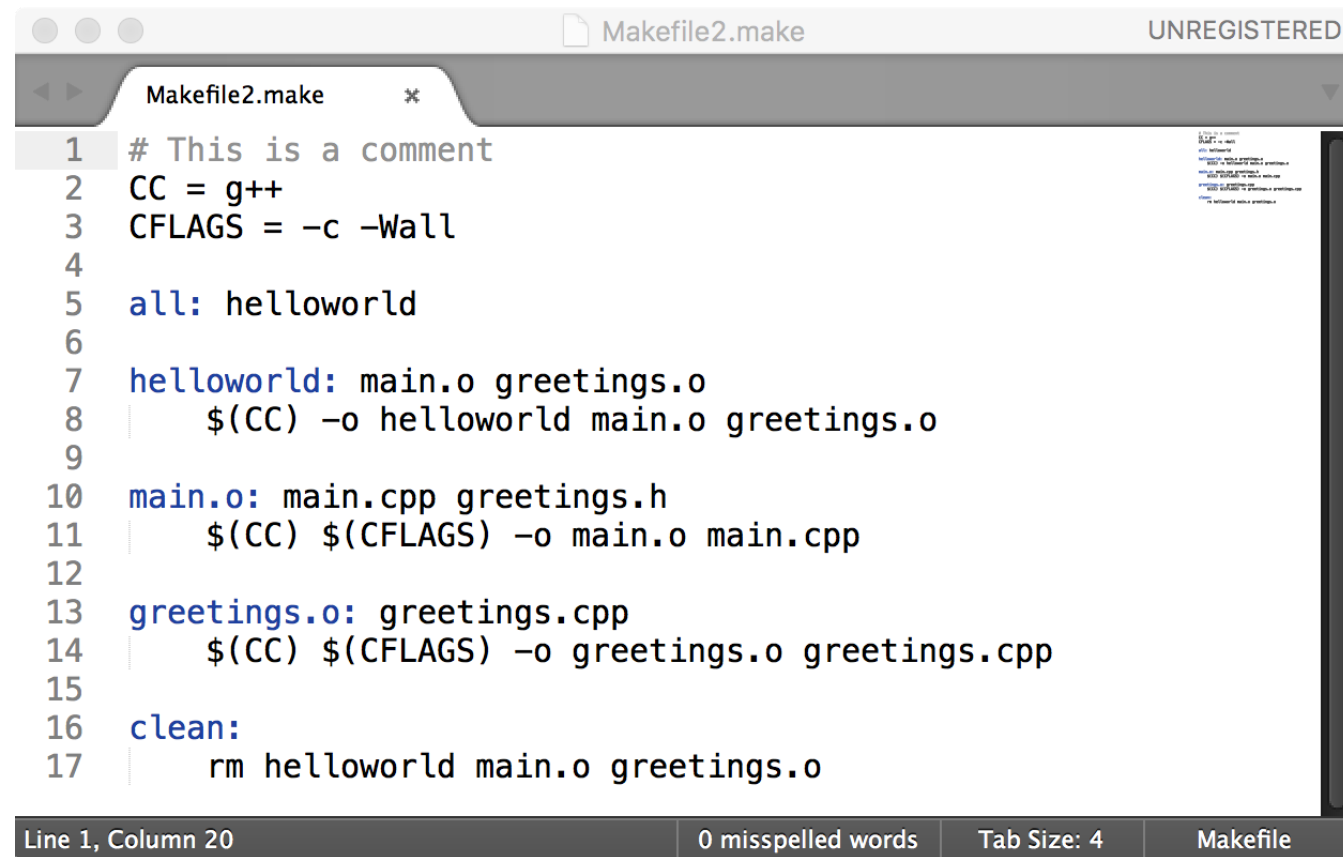


```
1 all : helloworld
2
3 helloworld: main.cpp greetings.cpp greetings.h
4             g++ main.cpp greetings.cpp -o helloworld
5
6 clean:
7       rm helloworld
```

The screenshot shows a text editor window titled 'Makefile' with a status bar at the bottom indicating 'Line 1, Column 1', '0 misspelled words', 'Tab Size: 4', and 'Makefile'. The editor contains a Makefile with three targets: 'all' which builds 'helloworld', 'helloworld' which builds from 'main.cpp', 'greetings.cpp', and 'greetings.h' using 'g++', and 'clean' which removes 'helloworld'.

Variables

- It is possible to use variables (*allows flexibility*)

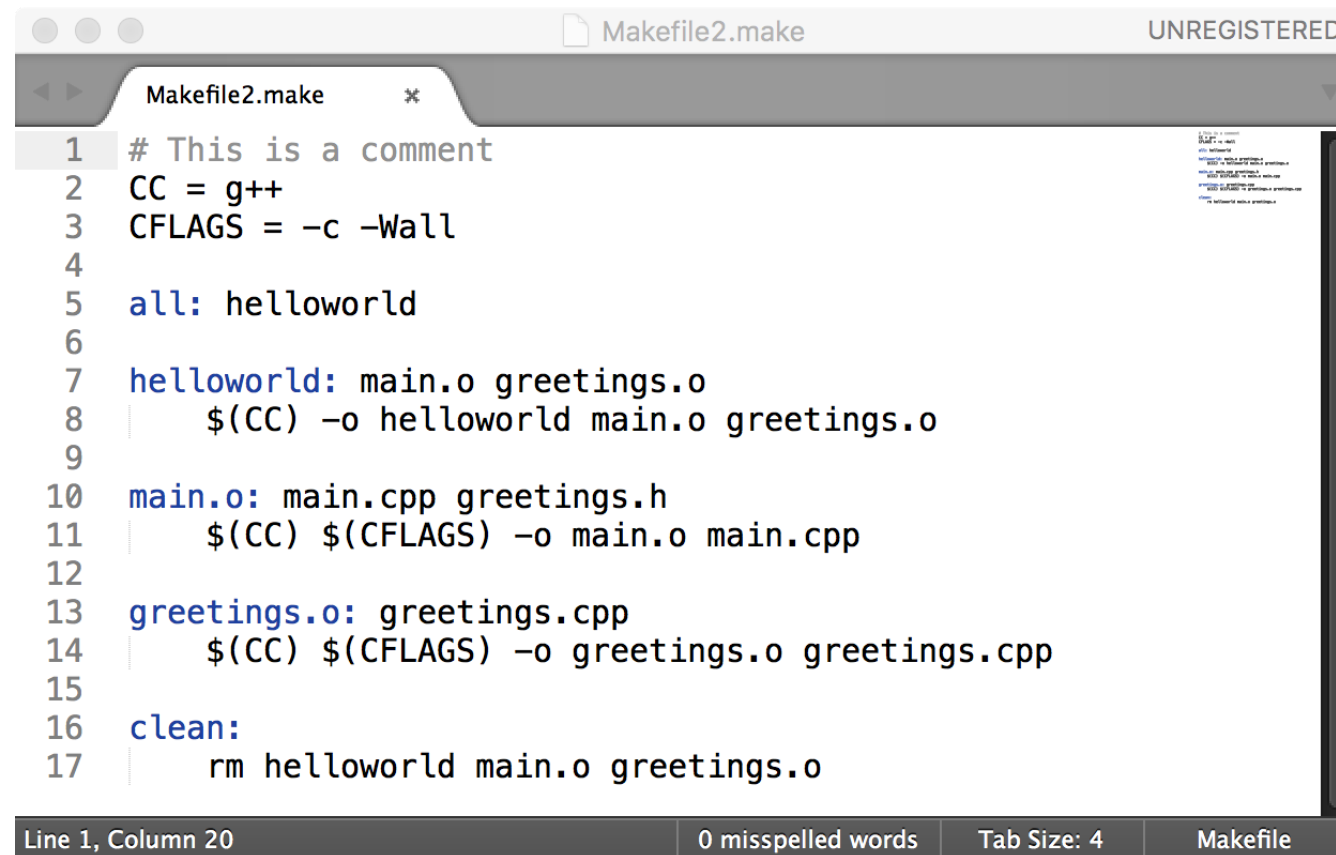


```
1 # This is a comment
2 CC = g++
3 CFLAGS = -c -Wall
4
5 all: helloworld
6
7 helloworld: main.o greetings.o
8 | $(CC) -o helloworld main.o greetings.o
9
10 main.o: main.cpp greetings.h
11 | $(CC) $(CFLAGS) -o main.o main.cpp
12
13 greetings.o: greetings.cpp
14 | $(CC) $(CFLAGS) -o greetings.o greetings.cpp
15
16 clean:
17 | rm helloworld main.o greetings.o
```

Line 1, Column 20 0 misspelled words Tab Size: 4 Makefile

Comments

- Use # to indicate a comment (*improves readability*)

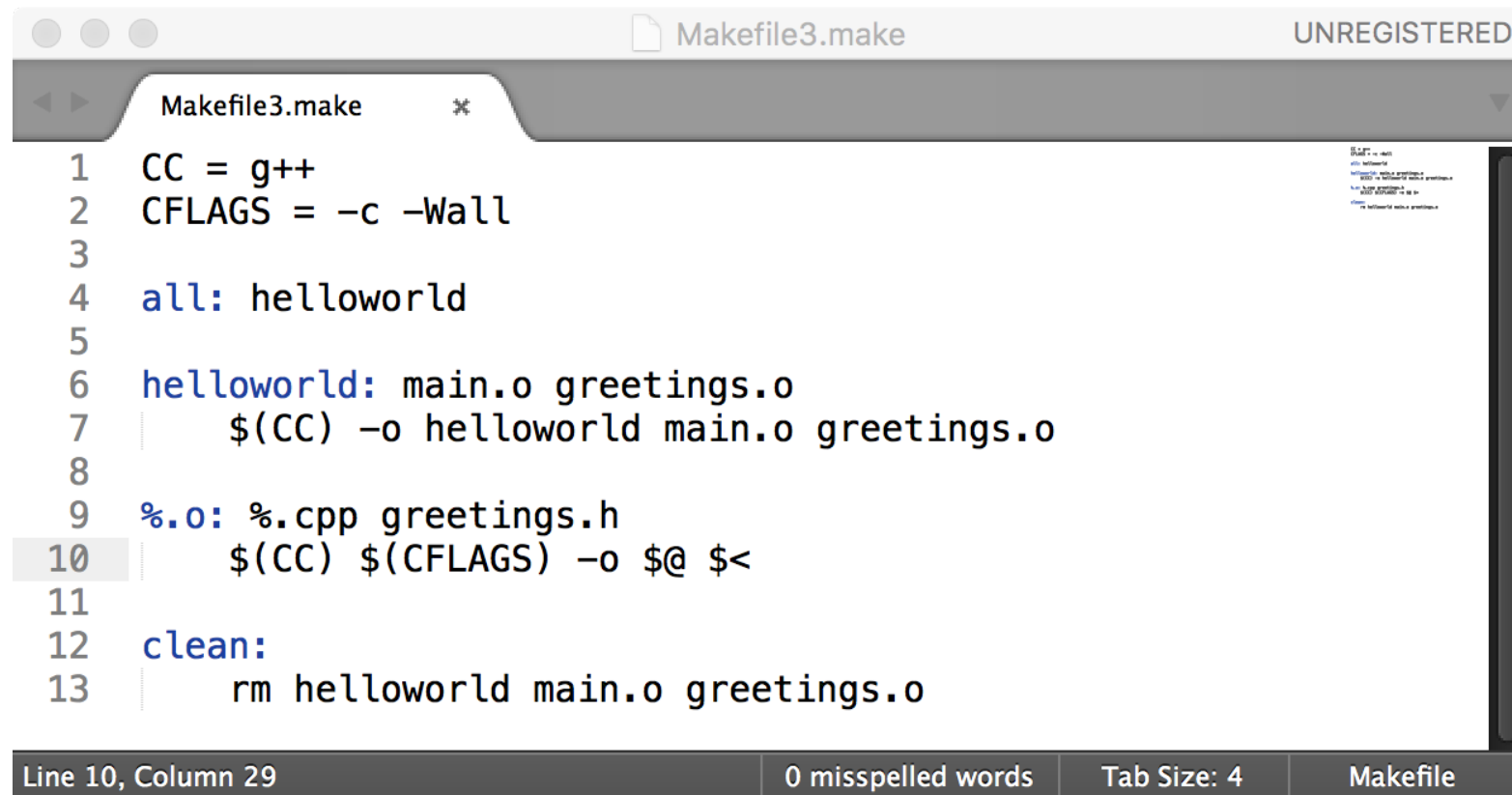


```
1 # This is a comment
2 CC = g++
3 CFLAGS = -c -Wall
4
5 all: helloworld
6
7 helloworld: main.o greetings.o
8 | $(CC) -o helloworld main.o greetings.o
9
10 main.o: main.cpp greetings.h
11 | $(CC) $(CFLAGS) -o main.o main.cpp
12
13 greetings.o: greetings.cpp
14 | $(CC) $(CFLAGS) -o greetings.o greetings.cpp
15
16 clean:
17 | rm helloworld main.o greetings.o
```

Line 1, Column 20 0 misspelled words Tab Size: 4 Makefile

Patterns

- It is possible to use patterns for building targets (*avoids repetitions*)

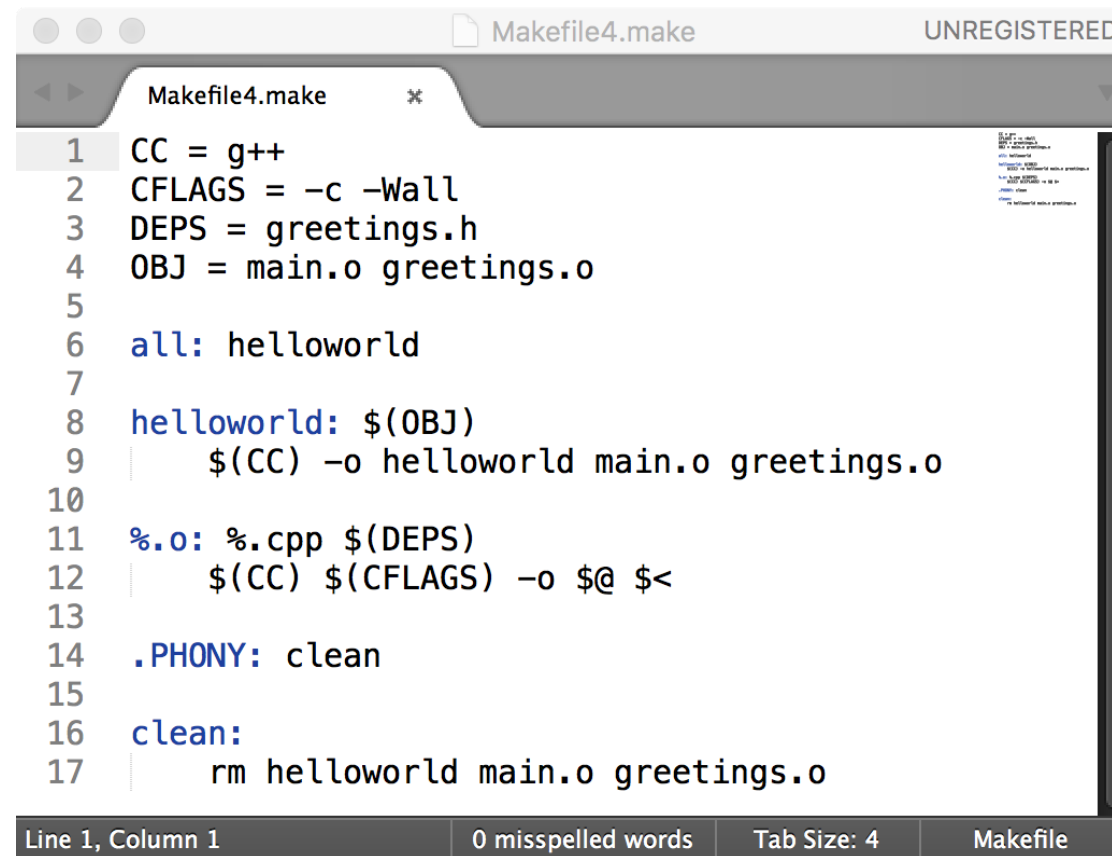


```
1 CC = g++
2 CFLAGS = -c -Wall
3
4 all: helloworld
5
6 helloworld: main.o greetings.o
7     $(CC) -o helloworld main.o greetings.o
8
9 %.o: %.cpp greetings.h
10    $(CC) $(CFLAGS) -o $@ $<
11
12 clean:
13    rm helloworld main.o greetings.o
```

Line 10, Column 29 0 misspelled words Tab Size: 4 Makefile

Specifying source

- It is possible to specify the source (*simplifies adding new source files*)



The screenshot shows a text editor window titled 'Makefile4.make' with a status bar indicating 'UNREGISTERED'. The editor contains a Makefile with the following content:

```
1 CC = g++
2 CFLAGS = -c -Wall
3 DEPS = greetings.h
4 OBJ = main.o greetings.o
5
6 all: helloworld
7
8 helloworld: $(OBJ)
9     $(CC) -o helloworld main.o greetings.o
10
11 %.o: %.cpp $(DEPS)
12     $(CC) $(CFLAGS) -o $@ $<
13
14 .PHONY: clean
15
16 clean:
17     rm helloworld main.o greetings.o
```

The status bar at the bottom shows 'Line 1, Column 1', '0 misspelled words', 'Tab Size: 4', and 'Makefile'.

Reference material

- GNU Make
<https://www.gnu.org/software/make/>
- Managing Projects with GNU Make, 3rd Edition by R. Macklenburg
<http://www.oreilly.com/openbook/make3/book/index.csp>