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 namesapce 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();
    re
}
```

main.cpp

```
#incl
using

void
```

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

```
#ifndef _greetings_h_
#define _greetings_h_
void say greetings();
#endif
```

greetings.h

```
> 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

dependencies

target

```
all : main.cpp greetings.cpp greetings.h
      g++ main.cpp greetings.cpp -o helloworld
```

system command

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

Makefile UNREGISTERED

Makefile

```
all : helloworld

helloworld: main.cpp greetings.cpp greetings.h
	g++ main.cpp greetings.cpp -o helloworld

clean:
	rm helloworld
```

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

A Basic Makefile

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

```
> make
```

Makefile UNREGISTERED

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
```

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

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

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
```

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
```

Makefile UNREGISTERED

Makefile

```
all : helloworld

helloworld: main.cpp greetings.cpp greetings.h
    g++ main.cpp greetings.cpp -o helloworld

clean:
    rm helloworld
```

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

Variables

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

Makefile2.make UNREGISTERED

Makefile2.make

```
# This is a comment
CC = g++
CFLAGS = -c -Wall

all: helloworld

helloworld: main.o greetings.o
    $(CC) -o helloworld main.o greetings.o

main.o: main.cpp greetings.h
    $(CC) $(CFLAGS) -o main.o main.cpp

greetings.o: greetings.cpp
    $(CC) $(CFLAGS) -o greetings.o greetings.cpp

clean:
    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*)

Makefile2.make UNREGISTERED

Makefile2.make

```
# This is a comment
CC = g++
CFLAGS = -c -Wall

all: helloworld

helloworld: main.o greetings.o
    $(CC) -o helloworld main.o greetings.o

main.o: main.cpp greetings.h
    $(CC) $(CFLAGS) -o main.o main.cpp

greetings.o: greetings.cpp
    $(CC) $(CFLAGS) -o greetings.o greetings.cpp

clean:
    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*)

Makefile3.make UNREGISTERED

Makefile3.make

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

all: helloworld

helloworld: main.o greetings.o
	$(CC) -o helloworld main.o greetings.o

%.o: %.cpp greetings.h
	$(CC) $(CFLAGS) -o $@ $<

clean:
	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*)

Makefile4.make UNREGISTERED

Makefile4.make

```
CC = g++
CFLAGS = -c -Wall
DEPS = greetings.h
OBJ = main.o greetings.o

all: helloworld

helloworld: $(OBJ)
    $(CC) -o helloworld main.o greetings.o

%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -o $@ $<

.PHONY: clean

clean:
    rm helloworld main.o greetings.o
```

Line 1, Column 1 | 0 misspelled words | Tab Size: 4 | 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