Makefiles

Joel Willoughby

COP 3503

Outline

Agenda

- Talk about some of the recurring problems from last lab
- Makefiles

Outline

Return values

- If you have a function that does some useful work and then returns a value you need (for instance, whether the work was successful), then when you want to use the result, don't call the function multiple times to check it.

```
1 set.insert("chocolate");
2 if(!set.insert("chocolate")){
3   cout << "Error, chocolate exists..." << endl;
4 }
```

Return values

- If you have a function that does some useful work and then returns a value you need (for instance, whether the work was successful), then when you want to use the result, don't call the function multiple times to check it.

```
1 set.insert("chocolate");
2 if(!set.insert("chocolate")){
3   cout << "Error, chocolate exists..." << endl;
4 }
```

Instead do:

```
1 bool success = set.insert("chocolate");
2 if(!success){
3   cout << "Error, chocolate exists..." << endl;
4 }
```

Returns necessary?

- A lot of times, functions don't need to return values at all.
- You should make these functions void
- For example, in our last lab, we needed to write a function that set a parameter instead of returning a value. It should be void.
- How do you return from a void function?

```
1 void fun_1(int & control){
2   //...
3   if(/*something bad happens*/){
4     control = 0;
5     return;
6   }
7   //...
8 }
```

Code after return

```
1 void fun_2(){
2   //Useful code
3   return;
4   int a = 0;
5 }
```

- Is line 4 ever executed?
- Don't put code after returns.

Outline

Review of Separate Compilation

- Idea is to put function and class declarations in something called a header (or .h) file. Then, you include that file in your .cpp where you define your functions (or classes).
- This allows you to separate your code into logical units that you can then compile separately.
- Allows you to have code reusability and portability

An Example

See website, done in lab

ifndef

- The ifndef directive should be added to all of your .h files to prevent multiple inclusion.
- Syntax looks likes:

```
1 #ifndef FILENAME_H_1234
2 #define FILENAME_H_1234
3
4 //Useful .h code here
5
6 #endif
```

Makefiles Motivation

- Make is a commad done from the command line (linux) that looks for and “executes” a makefile
- A makefile is basically just a sequence of commands that accomplish something. Normally, this is compilation of some large-ish (more than one file) project.
- It also keeps track of when dependencies change, and only compiles what *needs* to be compiled.
- Makes the compilation process for large projects so much simpler
- If you use an IDE, it automatically generates a makefile (it just hides it from you). You will write your own today.

A simple example

```
1 # Makefile example
2
3 all:
4     g++ main.cpp count.cpp -o count
```

- all is called a target. You can tell make to build specific targets. By default, it builds the first one in the file.
- The line following the targets *MUST BEGIN WITH A TAB CHARACTER*
- You don't just have to put g++ commands, any valid command line command will work fine. You can also do multiple lines.

Multiple Targets

```
1  all: count
2
3  count: main.o count.o
4      g++ -o count main.o count.o
5
6  main.o: main.cpp count.h
7      g++ -c main.cpp
8
9  count.o: count.cpp count.h
10     g++ -c count.cpp
11
12 clean:
13     rm -rf *.o
14     rm -rf count
```

Variables or macros

```
1  CXX = g++
2  CXXFLAGS = -c -Wall
3  LDFLAGS =
4
5  all: count
6
7  count: main.o count.o
8      $(CXX) $(LDFLAGS) -o count main.o count.o
9
10 main.o: main.cpp count.h
11     $(CXX) $(CXXFLAGS) main.cpp
12
13 count.o: count.cpp count.h
14     $(CXX) $(CXXFLAGS) count.cpp
15
16 clean:
17     rm -rf *.o count
```

Getting more fancy...

```
1  CXX = g++
2  CXXFLAGS = -c -Wall
3  LDFLAGS =
4  SRCS = main.cpp count.cpp
5  OBJS = $(SRCS:.cpp=.o)
6  DEPS = count.h
7  EXEC = count
8
9  all: $(EXEC)
10
11 $(EXEC): $(OBJS)
12     $(CXX) $(LDFLAGS) $(OBJS) -o $@
13
14 %.o: %.cpp $(DEPS)
15     $(CXX) $(CXXFLAGS) $<
16
17 clean:
18     rm -rf *.o count
```

Outline

Questions

???