Compiling and Creating a Makefile

Overall Theory

* this is a tool for compiling on a Linux/Unix server
* we are lazy
* we want a tool that will compile the many files we will have without typing much
  + you have to create the Makefile
  + you have to have a gameplan for it
* a makefile streamlines this task

Compiling a single file the manually (long way)

* using the g++(gcc) compiler
* from compile to run in two steps
  1. compile your code into a runnable output file
  2. run your output file
  3. program runs

|  |
| --- |
| Simple C++ single file setup |
| [slupoli@linux3 ~]$ g++ -o hello hello.cpp  [slupoli@linux3 ~]$ ./hello |

**Why use a Makefile**

* automated way of compiling
  + would have to compile every file!!
* saves compile time by only compiling those files that have changed
  + this is negligible if you have only a few files
  + but this is very helpful with real projects that have many files
* can “clean” up the mess from compiling
  + deletes old ??~ files and more

Creating a Makefile

* not just the number of files make a difference, but HOW they are linked together matters just as much
* the filename MUST be “Makefile”
  + capital M in Makefile
  + no extension
  + it can be something else, but then these notes would be off
  + our classes will use the same file name listed here
* overall gameplan
  + all .cpp(s) need to be compiled
    - since they include the .h(s), it will come too
  + name your results (or output) as “output”
  + compiling has an order
    - driver (main)
    - then sources
    - then

The mechanics of a Makefile

* there are many things to notice from the output and code
* output items of interest
  + notice driver than any other .cpp file is compiled
  + warning and errors may appear
    - which is what we want to happen if something is up
  + an OUTPUT file is created
    - we have to run that separately

|  |
| --- |
| If makefile runs smooth (w/ Airplane) |
| **[slupoli@linux3 Airplane]**$ make  g++ -Wall -c driver.cpp  g++ -Wall -c Airplane.cpp  Airplane.cpp: In constructor âAirplane::Airplane()â:  Airplane.cpp:32: warning: extended initializer lists only available with -std=c++0x or -std=gnu++0x  g++ -Wall driver.o Airplane.o -o output.out  **[slupoli@linux3 Airplane]**$ ./output.out  !!!Hello World!!!  100:100  100  0  <----------------->  crew->10  passenger->40  origin->Albany  destination->North Korea  <----------------->  crew->10  passenger->40  origin->Albany  destination->Aruba  [slupoli@linux3 Airplane]$ make clean |

Compiling/Running C++ using the makefile on GL

* This process will
  + compile project code
  + show any compiling errors you might have
  + run the program

|  |  |  |
| --- | --- | --- |
| Makefile procedure a 3 File Application | | |
| File Setup | | |
|  | | |
| Makefile | | |
| # do not need to include (or compile) .h files separately  Bottom up in “all”  all: driver.o Airplane.o  g++ -Wall driver.o Airplane.o -o output.out  Airplane.o : Airplane.cpp  g++ -Wall -c Airplane.cpp  driver.o: driver.cpp  g++ -Wall -c driver.cpp  # -f removes any error messages if the file is not present  clean:  rm -rf \*.o  rm -f \*.out  rm -f \*~ \*.h.gch \*# | | |
| Source files setup (look at includes) | | |
| header | Source | driver |
| #ifndef AIRPLANE\_H\_  #define AIRPLANE\_H\_  #include <string>  #include <iostream>  using namespace std;  … | #include <iostream>  #include <string>  using namespace std;  #include "Airplane.h" | #include <iostream>  #include <string>  using namespace std;  #include "Airplane.h" |

**C++ File setup for makefile compatibility**

* where and how the entire project includes is very important
* and so poorly documented
* this also is assumed that every STUDENT MADE file(s) for the project is within the same directory
  + example
    - <https://www.youtube.com/watch?v=PKh9suvS6bg>

|  |
| --- |
| Inheritance Example Makefile |
| File Setup |
|  |
| Makefile |
| all: driver.o Dog.o Animal.o  g++ -Wall driver.o Dog.o Animal.o -o output.out  Animal.o: Animal.cpp  g++ -Wall -c Animal.cpp  Dog.o: Dog.cpp  g++ -Wall -c Dog.cpp  driver.o: driver.cpp  g++ -Wall -c driver.cpp  # -f removes any error messages if the file is not present  clean:  rm -rf \*.o  rm -f \*.out  rm -f \*~ \*.h.gch \*# |

|  |  |  |
| --- | --- | --- |
| Source files setup | | |
| header (ANIMAL) | source (ANIMAL) |  |
| #ifndef ANIMAL\_H  #define ANIMAL\_H  #include <string>  using namespace std; | #include <iostream>  using namespace std;  #include "Animal.h" |  |
| header (DOG) | source (DOG) | driver |
| #ifndef DOG\_H  #define DOG\_H  #include <string>  using namespace std;  #include "Animal.h"  remember DOG inherits ANIMAL code | #include <iostream>  using namespace std;  #include "Dog.h" | #include <iostream>  using namespace std;  #include "Dog.h" |

Other tools embedded in a makefile

* clean
  + created by you the programmer
  + notice the command “tag” in the file
  + those given should help clean up many things that are left behind in the compile process
* selective compiling
  + say you made one change to the driver
  + instead of compiling the whole project, determine the command tag for that specific file
    - “driver.o” for driver files
    - “Dog.o” for source/header files of Dog
    - etc..

|  |
| --- |
| Command prompt &Output of various Makefile tools |
| [slupoli@linux3]$ make clean  rm -rf \*.o  rm -f \*.out  rm -f \*~ \*.h.gch \*#[  [slupoli@linux3]$ make Animal.o  g++ -Wall -c Animal.cpp  [slupoli@linux3]$ |

**Tabs, not spaces**

* If copying from the internet, make sure any indentions are with tabs not spaces
* this error appears if you don’t
  + Makefile:16: \*\*\* missing separator (did you mean TAB instead of 8 spaces?). Stop.

**Adding other features**

* we can add other features, or tags to help us in the overall process
* these tags do not have to have anything to do with the compiling process
  + much like clean

|  |
| --- |
| Running your project using a makefile |
| added to the Makefile |
| val: # to test for memory leaks  valgrind ./output  run:  ./output.out |
| how to activate |
| [slupoli@linux3 Airplane]$ make run  ./output.out  !!!Hello World!!!  100:100  100  0  <----------------->  … |

As an exercise, using only GL, create the object Lecturer. Create the appropriate data members, getters/setters, source code and driver shown below. Create the makefile that should compile this. I want to see the makefile “run”, “clean” and compile (make).

From your home directory:

1. mkdir CMSC341
2. mkdir CMSC341/makeExample
3. cd CMSC341/makeExample
4. cp –r /afs/umbc.edu/users/s/l/slupoli/pub/labCode341/makeFileLecturer .
5. ls (to see what files you copied)
6. The get to work!! The driver.cpp file is the ONLY file completed for you.

|  |  |  |
| --- | --- | --- |
| Lecturer.h | Lecturer.cpp | driver.cpp |
| firstName  lastName  age  salary  // constructors  // setters/getters | Source code for all in .h | Create ONE instance of a Lecturer.  Lecturer Park(“Park”, “John”, 112, 3000); |

Makefiles and Combined Header files

* this is a situation where a header file has BOTH source and the class
  + used to save space
* but this messes up the overall Makefile setup

|  |
| --- |
| No Source .cpp?? No problem |
| driver.out: BinarySearchTree.h driver.o  g++ -ansi -Wall BinarySearchTree.h driver.o -o driver.out  driver.o: driver.cpp  g++ -ansi -Wall -c driver.cpp  clean:  rm -f \*.o driver.out \*~ \*.gch  val: # to test for memory leaks  valgrind ./output  run:  ./driver.out |

Makefiles w/ Command Line Arguments (variables)

* the Driver (or whatever has the main) still needs to be able to accept the arguments
* but the makefile requires the use of variables within the RUN target
  + variables in the Makefile start with a $
  + give the variable an appropriate name(s)
  + ALL INPUT IS CONSIDERED STRINGS!!

|  |
| --- |
| makefile w/ an integer as arguments |
| Makefile |
| Driver.out: Driver.o  g++ -ansi -Wall Driver.o -o Driver.out  Driver.o: Driver.cpp  g++ -ansi -Wall -c Driver.cpp  clean:  rm -f \*.o Driver.out \*~ \*.gch  val: # each item with a $ is a variable NAME  valgrind ./Driver.out $(INT)  run: # each item with a $ is a variable NAME  ./Driver.out $(INT) |
| C++ Driver |
| #include <fstream>  #include <iostream>  using namespace std;  int main ( int argc, char \*argv[] )  {  if ( argc != 2 ) // argc should be 2 for correct execution  // We print argv[0] assuming it is the program name  cout<<"usage: "<< argv[0] <<" <INT> \n";  else  {  cout << argv[1] << endl;  cout << argv[2] << endl;  }  } |
| Linux Command |
| [slupoli@linux1]$ make run INT=23 |

|  |
| --- |
| makefile w/ many datatypes as arguments |
| Makefile |
| Driver.out: Driver.o  g++ -ansi -Wall Driver.o -o Driver.out  Driver.o: Driver.cpp  g++ -ansi -Wall -c Driver.cpp  clean:  rm -f \*.o Driver.out \*~ \*.gch  val: # checking for memory leaks  valgrind ./Driver.out $(INT) $(FLOAT) $(STRING) $(LONG\_STRING)  run: # each item with a $ is a variable NAME  ./Driver.out $(INT) $(FLOAT) $(STRING) $(LONG\_STRING) |
| C++ Driver |
| #include <fstream>  #include <iostream>  #include <cstdlib> // or sometimes <stdlib>  using namespace std;  int main ( int argc, char \*argv[] )  {  if ( argc != 5 ) // argc should be 5 for correct execution  // We print argv[0] assuming it is the program name  cout<<"usage: "<< argv[0] <<" <INT> <FLOAT> <STRING> <LONG STRING>\n";  else  {  cout << "[0] " << argv[0] << endl;  cout << "[1] " << argv[1] << endl;  cout << "[2] " << argv[2] << endl;  cout << "[3] " << argv[3] << endl;  cout << "[4] " << argv[4] << endl;  cout << "[5] " << argv[5] << endl;  int score = atoi(argv[0]);  float temperature = atoi(argv[1]);  }  } |
| Linux Command |
| [slupoli@linux1]$ **make run** INT=23 FLOAT=23.3 STRING="Lupoli" LONG\_STRING="'Needs a Life'"  ./Driver.out 23 23.3 Lupoli 'Needs a life'  [0] ./Driver.out (notice NOT **make** or **run**!!)  [1] 23  [2] 23.3  [3] Lupoli  Needs a life |

Create a new directory “ExMakefiles”. Inside create a driver file that will loop and display “Hello” as many times the command line **variable** COUNT asks it to. Your makefile will also be different and very simple since only one file.

**Don’t forget to “make” first (compile), then make run…**

[**Answerurl:**](https://www.youtube.com/watch?v=Wg34KpUxyts)

If time permits: 3 inputs, (nums), 1 text => 222-99-9999-ZZZZ

Answerb:

**Makefiles with Macros**

* think of them as constants
* optional
* useful when a lot of repeating
* thanks to Frank Zastawnik for the simple example

|  |
| --- |
| Makefiles with Flags |
| CXX = g++  CXXFLAGS = -ansi -Wall -g  OBJS = Driver.o Graph.o Util.o  all: $(OBJS)  $(CXX) $(CXXFLAGS) $(OBJS) -o Driver.out    Driver.o: Driver.cpp  $(CXX) $(CXXFLAGS) -c Driver.cpp    Graph.o: Graph.h Graph.cpp Util.h  $(CXX) $(CXXFLAGS) -c Graph.cpp    Util.o: Util.h Util.cpp  $(CXX) $(CXXFLAGS) -c Util.cpp    clean:  rm \*.o\*  val:  valgrind ./Driver.out $(FILE)    run:  ./Driver.out $(FILE) |

**Solutions**

|  |
| --- |
| Multiple Command Line Arguments |
| Makefile |
|  |
| Driver |
| #include <iostream>  #include <stdlib.h>  using namespace std;  int main(int argc, char \*argv[]){  if(argc != 5){  cout << "Error: wrong number of arguments" << endl;  } else {  cout << atoi(argv[1]) << "-" << atoi(argv[2]) << "-" << atoi(argv[3]) << argv[4] << endl;  }  return 0;  } |

**Sources:**

<http://myweb.stedwards.edu/laurab/help/makefilehelp.html>

<http://www.cs.umd.edu/class/fall2002/cmsc214/Tutorial/makefile.html>

<http://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html>

<http://faculty.kutztown.edu/spiegel/Makefile/Makefile.htm>