CSCI-2270 Data Structures Recitation 13// Instructors: Hoenigman/Zagrodzki/Zietz

Learning Objectives:

- makefile
- gdb

1 Makefile

Make is Unix utility that is designed to start execution of a makefile. A makefile is a special file, containing shell commands, that you create and name makefile (or Makefile depending upon the system). While in the directory containing this makefile, you will type **make** and the commands in the makefile will be executed. If you create more than one makefile, be certain you are in the correct directory before typing make.

Make keeps track of the last time files (normally object files) were updated and only updates those files which are required (ones containing changes) to keep the source file up-to-date. If you have a large program with many source and/or header files, when you change a file on which others depend, you must recompile all the dependent files. Without a makefile, this is an extremely time-consuming task.

1.1 Contents

Here we will discuss basic contents of a makefile.

1.1.1 Variables

Makefile may contain variable. For example-
$CFLAGS = -g - Wall$
$CC = g++$
To use the variable we need to type $(var_name).

1.2 Rules

Rules are in following format-
$< target >:< dependency files separated by space >$
$< tab > command to generate the target file.$
For example you have three files- graph.h, graph.cpp and main.cpp. You want to generate the .out file. So the corresponding makefile will contain-

```
main.o:  graph.h graph.cpp main.cpp
   g++ -std=c++11 -g -Wall graph.cpp main.cpp -o main.o
```

Using the variable from previous section

```
main.o:  graph.h graph.cpp main.cpp
   $(CC) -std=c++11 $(CFLAGS) graph.cpp main.cpp -o main.o
```

2 GDB

2.1 Installing GDB

2.1.1 MAC

Update your homebrew and then run the command 'brew install gdb'. For more detail refer to `http://panks.me/posts/2013/11/install-gdb-on-os-x-mavericks-from-source/`. For setting up GDB on Mac OS Sierra/High Sierra follow the instruction from- `https://gist.github.com/danisfermi/17d6c0078a2fd4c6ee818c954d2de13c`.

2.1.2 unix/Linux

Generally unix distributions come with GDB. However you can install it using the following commands from terminal-

- sudo apt-get update
- sudo apt-get install gdb

For more details refer to `http://www.gdbtutorial.com/tutorial/how-install-gdb`.

2.1.3 windows

I assume all of you have minGw installed. So browse to the installation folder `C:\MinGW\bin`. Then run the command- `mingw-get.exe install gdb`.
You can do it using GUI as well. You need to open the MinGW installation Manager. Then select the GDB from the list of packages. Refer to this youtube link- `https://www.youtube.com/watch?v=BoB-4O3ZyhQ`.

2.2 How to debug

In general you might be using the following command to compile your code-
`g++ -std=c++11 prog.cpp -o a.out`.
Now to enable your program to run with debugger you need to put `-g` option. So the command becomes-
`g++ -std=c++11 -g prog.cpp -o a.out`.
You can start debugger on your program by one of the two ways-

- type the following command `gdb a.out`
- first give `gdb` command and then `file a.out`

Once we have given the file to run the debugger use the command `run`.

- **Break**
  We need debugger to stop at certain point in the code so that we can investigate the program. To set a breakpoint we will use the command `break`.

  ```
  break prog.cpp:12
  ```

  In this example we are setting a break point in file `prog.cpp` at line number 12. If we are interested in a particular function we can set break point at that function and the debugger will stop every time the function is called.

  ```
  break myfunc
  ```

- **Continue**
  To move on to next break point we can use the command `continue`.

- **step** and **next**
  To proceed by single-step you can either use **step** or **next**. However there is a subtle difference between `step` and `next`. If your next line of code is a function call `next` will consider it as a single instruction and will execute the function all at once. On the other hand `step` will take you though lines of the function. So `step` gives more fine-grained control than `next`.

- **print** and **display**
  To print a value of a variable we can use `print` command.

  ```
  print my_var
  ```

  `print` will print the value only once. If you want to print the value each time your are in the scope where the variable is defined you can use the command `display`.

- **watch** Whereas breakpoints interrupt the program at a particular line or function, watchpoints act on variables. They pause the program whenever a watched variable's value is modified.

  ```
  watch my_var
  ```

NOTE: Each time you make any change in the code you need to stop the gdb and recompile the program to generate the updated executable (.out). Now you need to run the gdb with this new `.out` file.

Acknowledgement
I find slides from this site(`https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf`) to be useful. I have used these slides for creating this write up.

3 Exercise

Download the files provided. `test_int_array.cpp` contains the `main` function.
`terrible_dynamic_size_array_unsorted.h` and `terrible_dynamic_size_array_unsorted.cpp` has declaration and definition of a array data structure and its functionality. This code has bugs. Try to find out bugs using gdb. On finding one bug fix that bug, keep a comment in that line about the bug and move on to find the next bug. Note that you need to fix the bug so that you can carry on the code execution to find the next bug.