EASTERN WASHINGTON UNIVERSITY
start something big

Make and GDB

Computer Science Department
Eastern Washington University
Yun Tian (Tony) Ph.D.

Today

- Make Tool on Unix/Linux
- GDB debug tool

Header files

- Usually, we put all **related** function prototypes into a separate .h file, called header file.
  - int send(char * buff) in sender.h file
- If we need to call **send** function in another c file, we just include the header file on top that c file.

  #include “sender.h”

Simple Example

- hellomake.c

```
#include “hellomake.h”
int main()
{
   // call a function in another file
   myPrintHelloMake();
    return(0);
 }
```

hellofunc.c

```
#include <stdio.h>

void myPrintHelloMake(void)
{
   printf("Hello makefiles!\n");
   return;
}
```

hellomake.h

```
/* example include file */
 void myPrintHelloMake(void);
```

Make Tool

- Normally, you would compile this collection of code by executing the following command:

```
gcc -o hellomake hellomake.c hellofunc.c -I.
```

- This compiles the two .c files and names the executable hellomake.
- The **-I.** is included so that gcc will look in the current directory (.) for the include file hellomake.h.

Make Tool

- **Without** a makefile, the typical approach to the test/modify/debug cycle is to use the up arrow in a terminal to go back to your last compile command.
  - Although you don't have to type it each time,
  - Especially once you've added a few more .c files to the mix.

Two Disadvantages without makefile

- First, if you lose the compile command or switch computers you have to retype it from scratch,
  - which is inefficient at best.
- Second, if you are only making changes to one .c file,
  - Recompiling all of them every time is also time-consuming and inefficient.

makefile 1

- A simple makefile

  makefile

```
hellomake: hellomake.c hellofunc.c
        gcc -o hellomake hellomake.c hellofunc.c -I.
```

- If you put this rule into a file called **Makefile** or **makefile** and then type make on the command line,
  - It will execute the compile command as you have written it in the makefile.
- Furthermore, by putting the list of files on which the command depends in the first line after the **:**,
  - make knows that the rule hellomake needs to be executed if **any of those files change**.

makefile 1

- One very important thing to note is that
  - there is a **tab** before the gcc command in the makefile. There **must be a tab** at the beginning of any command.

Issues in makefile 1

- We solved the first issue, that we don't need to retype the command to compile.
- However, the system is still not being efficient in terms of compiling only the latest changes.
- If we change either hellomake.c or hellofunc.c, both of them are recompiled.

makefile 2

makefile

```
CC=gcc
CFLAGS=-I.
DEPS = hellomake.h

hellomake: hellomake.o hellofunc.o
	gcc -o hellomake hellomake.o hellofunc.o -I.

hellomake.o: hellomake.c $(DEPS)
	$(CC) -c -o hellomake.o hellomake.c

hellofunc.o: hellofunc.c $(DEPS)
	$(CC) -c -o hellofunc.o hellofunc.c

clean:
	rm -r *.o hellomake
```

The makefile compiles only files that are modified since last make run.

makefile 2

makefile

```
CC=gcc
CFLAGS=-I.
DEPS = hellomake.h

%.o: %.c  $(DEPS)
		$(CC) -c -o $@ $< $(CFLAGS)
hellomake: hellomake.o  hellofunc.o
		gcc -o hellomake hellomake.o hellofunc.o -I.
```

- This file first creates the macro CC, CFLAGS, and DEPS.
  - DEPS is the set of .h files on which the .c files depend.
- Then we define a rule that applies to all files ending in the .o suffix.
- The rule says that the .o file depends upon the .c version of the file and the .h files included in the DEPS macro.

make file 2

- The rule then says that to generate the .o file, make needs to compile the .c file using the compiler defined in the CC macro.
- The -c flag says to generate the object file,
- The -o $@ says to put the output of the compilation in the file named on the left side of the :
- The $< is the first item in the dependencies list, and the CFLAGS macro is defined as above.

Demo of make file 2

- Now If we only modify hellofunc.c,
- make will only re-compile hellofunc.c, without recompiling all of .c file.

Demo of GDB tool

- Basic Essential Commands with GDB

Demo of valgrind

- valgrind can check memory leak and correctness issues.
  - To use this tool, you have to install it or you go to cslinux machine.
- **If you normally run your program like this:**

  ./myprog arg1 arg2
- **With valgrind you run your program,**

  valgrind ./myprog arg1 arg2
  - It shows you some memory on heap you allocated, but forget to deallocate them if any.

Next Class

- 2D array and Double pointers