CSE130: Principles of Computer Systems Design

Teaching Assistant: Nilufar Ferdous

C program: hello.c

```
#include <stdio.h>

 int main(void)

{

   printf("Hello, World!\n");

}
```

Compile a basic C program using Clang

```
$ clang hello.c -o hello
```

- Hello.c: source file
- -o: compiler flag used to name the output binary file

C Program using GetString() : hello.c

```
#include <cs50.h>

#include <stdio.h>

 int main(void)

{   printf("What is your name?");

    // Get text input from user:

    string name = GetString();

    // Use user input in output striing:

    printf("Hello, %s\n", name);

}
```

Compile

```
$ clang hello.c -o hello
```

Error

Terminal Output from Command:

```
/tmp/hello-E2TvwD.o: In function `main':

hello.c:(.text+0x22): undefined reference to `GetString'

clang: error: linker command failed with exit code 1

    (use -v to see invocation)
```

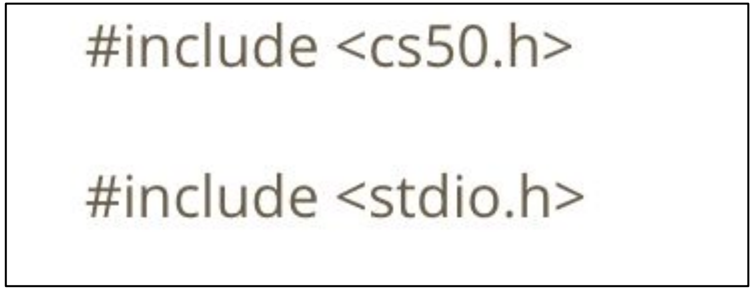
How to Solve ?

- From the terminal output, we can see that the compiler cannot find the GetString() method
- There was an issue with the linker.

Solution:

- We can add some additional arguments to our clang command to tell Clang what files to link:

```
$ clang hello.c -o hello -lcs50
```



```
#include <cs50.h>

#include <stdio.h>
```

-l Flag : -l flag followed by the name of the library we need to include, we have told Clang to link to the cs50 library.

Handling Errors and Warnings

```
clang hello.c -g -Wall -o hello -lcs50
```

-g Flag: tells the compiler to include debugging information in the output files

-Wall Flag: turns on most of the various compiler warnings in Clang

Introduction to Makefile

Makefile:

"The make utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them."

- Make utility utilizes structured information contained in a makefile in order to properly compile and link a program.
- is placed in the source directory for your project.

Basic Structure of Makefile

```
# the compiler to use
CC = clang

# compiler flags:
#  -g    adds debugging information to the executable file
#  -Wall turns on most, but not all, compiler warnings
CFLAGS  = -g -Wall

#files to link:
LFLAGS = -lcs50

# the name to use for both the target source file, and the output file:
TARGET = hello

all: $(TARGET)

$(TARGET): $(TARGET).c
	$(CC) $(CFLAGS) -o $(TARGET) $(TARGET).c $(LFLAGS)
```

```
clang hello.c -g -Wall -o hello -lcs50
```

Rules for Makefile

- In a makefile, lines preceded with a hash symbol are comments
- it is critical that the <TAB> on the third line is actually a tab character. Using Make, all actual commands must be preceded by a tab.

Compile using Makefile

```
$ make
```

Makefile using Clang++: SOURCES & INCLUDES

```
# List the sources, separated by spaces, for your program. 'SOURCES=foo.cpp bar.cpp', for example. Do not include anything except sources files that you will compile!

SOURCES=dog.cpp bar.cpp

INCLUDES=$(wildcard *.h)

# This is the name of the executable that this makefile will produce.

TARGET=dog
```

Makefile using Clang++ :_CXXFLAGS

```
# Define the compiler flags for compiling your code. You may wish to change these for debugging or optimization purposes. We suggest, that when you submit your code, you use the flags _submit_CXXFLAGS below:

_submit_CXXFLAGS=-std=gnu++11 -Wall -Wextra -Wpedantic -Wshadow -g -O2

# These are the flags that you should submit your code with. We will make sure that your code is compiled with these flags when we grade it.

# We're using clang as our C++ compiler.

CXX=clang++
```

Objects

# Generate a list of object files from the source files. If SOURCES is 'foo.cpp bar.cpp', then OBJECTS will contain 'foo.o bar.o'. These are the files that will be produced by the compiler,one for each compilation unit.

```
 SOURCES=dog.cpp bar.cpp
```

```
OBJECTS=$(SOURCES:.cpp=.o) //dog.o bar.o
```

DEPENDENCIES: DEPS

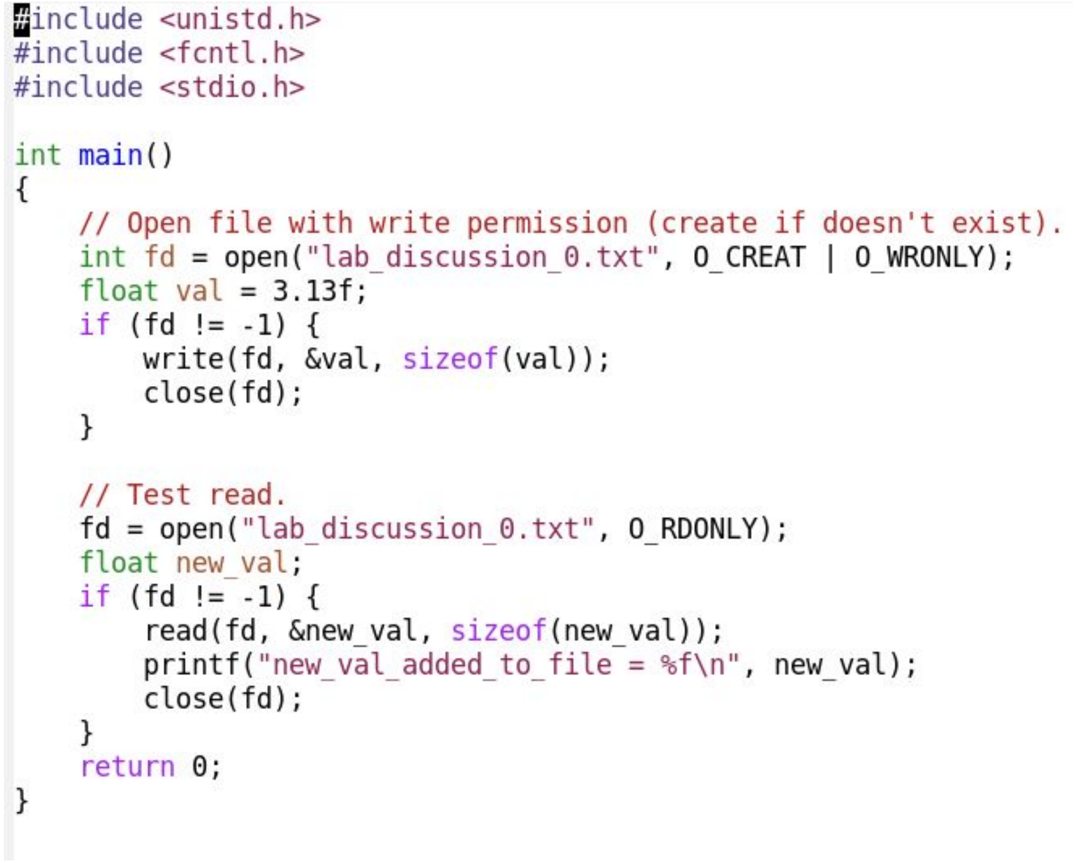
Dependency files will be .d files and will contain a list of _dependencies_ for each source file. You can specify these manually, like 'dog.cpp: dog.h bar.h' if you want, but the method in this makefile generates them as:

```
DEPS=$(SOURCES:.cpp=.d) // dog.cpp: dog.h bar.h
```

Clean

```
# provide a default target, with the standard name 'all'.

all: $(TARGET)

clean:

	-rm $(DEPS) $(OBJECTS) //dependency ( .d) files) and object (.o) files

spotless: clean

	-rm $(TARGET)
```

Reading & Writing Files using System Calls:



```c
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main()
{
    // Open file with write permission (create if doesn't exist).
    int fd = open("lab_discussion_0.txt", O_CREAT | O_WRONLY);
    float val = 3.13f;
    if (fd != -1) {
        write(fd, &val, sizeof(val));
        close(fd);
    }

    // Test read.
    fd = open("lab_discussion_0.txt", O_RDONLY);
    float new_val;
    if (fd != -1) {
        read(fd, &new_val, sizeof(new_val));
        printf("new_val_added_to_file = %f\n", new_val);
        close(fd);
    }
    return 0;
}
```

System Calls needed:

socket,bind, listen, accept, connect, send, recv, open, read, write, close, warn

Next week:

Socket Programming using these concepts

Question?

thank you