CS350 Lab0

Table of Contents

- Useful Stuff
  - Command line arguments
  - Header files
  - GCC
  - Make
  - GDB
- Warmup Lab Activity

Useful Stuff You Should Know

Command Line Arguments

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[])
{
int val = atoi(argv[1]);
printf("%d\n", val*10);
return 0;
}
```

```
int main(int argc, char* argv[]) { ... }
```

argc – The number of arguments

argv – Each argument as a C-string

Note: argv[0] is the command used to start the program

```
$ ./a.out 1 "Hello world"
```

argv[0] = “a.out”

argv[1] = “1”

argv[2] = “Hello world”

Header Files

- Contains function declarations and macro definitions
- Shared between several source files
- *Should not* contain variable definitions or function code
- Header file is included in the program by using the C preprocessing directive **#include.**
  e.g. #include <stdio.h>
- Including a header file is equal to copying the content of the header file

Header File Errors

```c
//foo.h
struct foo {
        int i;
        float f;
};
```

```c
//bar.h
#include “foo.h”
void bar(struct foo* f);
```

```c
//main.c
#include “foo.h”
#include “bar.h”
int main() {
...
}
```

```
$ gcc -c main.c
In file included from bar.h:1:0, from main.c:2:
foo.h:1:8: error: redefinition of ‘struct foo’
struct foo {
       ^
In file included from main.c:1:0:
foo.h:1:8: note: originally
defined here
struct foo {
       ^
```

#include Guards

```
//foo.h
#ifndef FOO_H
#define FOO_H
struct foo {
int i;
float f;
};
#endif
```

- The first inclusion of "foo.h" causes the macro `FOO_H` to be defined.
- Second inclusion of foo.h will cause the preprocessor to skip down to the `#endif`, thus avoiding the second definition of struct foo.

GCC

- GNU Compiler Collection
- Originally only compiled C, now compiles C, C++, and many more languages
- Gcc performs two operations:

a) compiling: convert source to object code

b) linking : combining the necessary object code files together into one complete executable.

To compile a single file, file1.c:

gcc -c file1.c (creates object file: file1.o)

To link your files to an executable:

gcc -o output file1.c file2.c (creates object file: output.o)

Makefile

- It is basically a sequence of commands which describes how the program can be constructed from source files.
- Consists of a set of targets, dependencies and rules.
- Target is the output file that is created or updated. Target depends upon a set of files (source files, header files) which are mentioned in Dependency List. Rules are the necessary commands to create the target file by using Dependency List.

```
target1:file1 file2 … fileN
          [tab]     command1
          [tab]     command2
```

file1, file2, fileN are the dependencies for the target1.

command1, command2 are the rules.

e.g.

```
main.o: main.c
        gcc -c main.c
```

Note: The line with gcc is prefixed by a tab, not whitespaces; else it will throw a "missing separator" error. You may list as many commands as required, as long as you don't leave a blank line and all commands start with a tab.

Makefile

An example of makefile:

```
mymake:    sample.o myhead.o myfile.o
        gcc -o samoutput sample.o myhead.o myfile.o

sample.o:   sample.c myhead.h myfile.h
        gcc -c sample.c

myhead.o:  myhead.c myhead.h
        gcc -c myhead.c

myfile.o:     myfile.c myfile.h
        gcc -c myfile.c
```

GDB - Debugger

- Command-line tool for analyzing and debugging a program to find bugs/errors
- Can be used for breakpoints, stepping through code, printing values, finding segmentation faults, etc.

Warmup Lab Activity

Q1: Write a program to implement Recursive Fibonacci Sequence:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, …

Fibonacci Definition: F(n) denotes Fibonacci function where n>0

F(1) = 1

F(2) = 1

F(n) = F(n-1) + F(n-2)

Warmup Lab Activity

- Create three files
- main.c - Takes argument, calls fib, prints result
- fib.h - Contains recursive fib declaration
- fib.c - Contains recursive fib implementation

- Use a makefile to compile both .c files into .o files and then compile the .o files into one executable: fib

- Example:

```
$ ./fib 10
55
```

Warmup Lab Activity

Q2: Write a program to implement a queue data structure using a dynamic array of size ‘n’.

Queue is a FIFO data structure i.e. the least recently added item is removed first.

Warmup Lab Activity

- Create one file
  - queue.c - Takes an integer n as an argument. n is the size of the queue.
  - Provide an interface to insert and remove numbers from the queue.
  - Initially the queue is supposed to be empty.
- Use the same makefile to compile this and create an executable: queue
- Example:

```
$ ./queue 4
$ 1. Insert
   2.Remove
   3.Exit
```

This will create a queue of size 4 i.e. it can store maximum 4 numbers at a time. It shows an interface (any basic/simple command line interface) that allows user to insert and remove numbers from the queue.