CIS2520

Lab#1: If you build it…

Week 1: Sept 11th, 2017

# Overview

In this lab session, students will review c programming material and makefiles. Students will create a makefile to handle multiple c files and programs.

# Learning Outcomes

Upon successful completion of this lab students should be able to:

* Create a makefile that compiles their main and test main
* Use makefile variables, targets, and commands
* Develop a makefile as a starting point for future labs and assignments

# Review Material

* Makefile
* C Review
* Pointers

# Introduction

A Makefile is a text file containing a simple set of instructions to run your program, it indicates which files should be compiled and defines common commands. Makefiles are a fundamental component that must be integrated into the development of your labs and assignments throughout the course. Please review the material in CourseLink for an introduction to MakeFiles.

After weekly lab session finish, related lab material will be available on GitHub. During the lab, you will see references such as (Lab 1 – GitHub Stage3) which indicate which lab and section you will find the corresponding code.

# Getting Started

## Getting to know your compiler

To make sure everyone is up and running with their compiler start with the following code snippet. Put this code in a file called ‘main.c’ and compile it with GCC.



### Questions:

1. What happens when you compile this code with –std=c89 versus –std=c99 flags?
2. What is the main difference between c89 and c99 options?
3. What does gcc –c main.c do? Describe the result.
4. Review and give a brief description of the following compiler flags: pedantic, Wall, pedantic-errors.

# Building a Project

This lab section will build a project in which students will create a makefile that creates two programs one as a main and another as a testing program.

## Person Header File

Create a header file called ‘person.h’ with a struct called Person it should have two members: int age; and char name[30];

Declare the following functions in the person.h header file.

These comments are ok but could be improved, they tell us basic information but it would much more useful and professional to write them in Doxygen format. **Doxygen will be the comment standard for this course.**

Task: Rewrite these in Doxygen format, check out Doxygen Wikipedia page for examples. NOTE: you don’t have to compile into Doxygen documents just use the format.

## Person Source File

Create the ‘person.c’ implementation for the above functions.

## Main and Test Main Source Files

Create two files ‘main.c’ and ‘testMain.c’ each with their own main. Include a print statement at the top of each main to indicate which main is running. Write code that will create two people using the Person structure, per file, use unique information for each person and write that information to the console screen before exiting the main.

## Building your makefile

Create a makefile that builds two separate programs. The first target should be called ‘program’ which uses ‘main.c’ and a second target called ‘test’ using ‘testMain.c’. You should be able to build either program by calling: ‘make program’ or ‘make test’. If you’re unsure of how to do this please review the makefile review on CourseLink. (Lab 1 – GitHub Stage1)

In your makefile, build a separate rule to clear the screen calls ‘cls’. Create a 2nd rule that will display your name and a short description of the program. Make sure only the information requested are displayed to the screen. ( Lab 1 – GitHub Stage2)

### Extending your makefile

In your makefile, create two variables called ‘CC’ and ‘CFLAGS’, store the compiler name and compiler flags with these variables respectively. Use the variables throughout your makefile instead of GCC and the flags.

## Building a standard Project

If you’re working in one directory lets clean this up and build a proper project folder structure. Programmers will expect software projects to follow a basic structure this helps everyone quickly navigate the project and keeps it organized.

To get started, remove the old program and testing executables that were built. The root level (top level) of your project should only ever contain a makefile, readme file, and several folders.

Create a file called readme with your name and a description of the program create the following folders bin, src, includes, docs, and assets.

The bin folder is for all programs or executables, src holds your source files, includes is for c-header files, docs is for project documentation except the readme which stays at the top level, and the assets folder which includes any data files. The make file should stay at the top level and should always have a default option that builds the entire project. The make default is always the first rule in your make file.

Move all project files into their appropriate folders.

Adapt your makefile to work exactly as before within the new folder structure. Your program and testing programs should be compiled into the bin folder. (Lab 1 – GitHub Stage3)

Try and see if you can reduce the repetitiveness of the makefile with other variables.

Investigate what the following two cmake calls do:

$(wildcard \*.c)

$@

## Start Testing

Your testing file should include two people similar to your main.c. We will cover testing in more depth in lab 2. For now, think about when your program might crash what kind of input might cause crashes. Write code to protect your function against that case.