# Assignment 1: Checksum

CSE30: Computer Organization and Systems Fall 2021

Instructors: Bryan Chin and Keith Muller

Due: Sunday October 3rd, 2021 @ 5:00PM

This is a **mini programming assignment**. This is meant to be a gentle introduction to programming you will do in this class. **Do NOT use this as a gauge for the difficulty of future assignments nor the amount of time required to complete future assignments. FUTURE ASSIGNMENTS WILL TAKE MORE TIME. Since this is a mini programming assignment, this assignment is weighted half as much as those in the future.**

# Table of Contents



# Learning Goals

- Learn to write C programs that can do simple character I/O (input/output).
- Be able to use development tools on the Raspberry Pi cluster.
- Learn (or relearn) how to create C programs with Makefile and source files.

# Assignment Overview

Over the summer, you have been playing on a public Minecraft server with a large player base. You've been able to make friends with other people on the server, and have recently started a building project to create a town together! However, because of school starting up again, you and your friends have been unable to find times where you can play together. So in order to communicate, the group has been using signs to leave messages for each other.

Unfortunately, some other random players on the server have been messing with your group and have been leaving misleading messages all over the place! To counteract this, the town decides to create a method to determine whether a message is genuine by ending messages with a simple checksum signature (that hopefully the random players can't crack)!

# Getting Started

## Developing Your Program

For this class, you **MUST** compile and run your programs on the **pi-cluster**. To access the server, you will need your `cs30fa21cxx` student account, and know how to SSH into the cluster.

**Need help or instructions? See Edstem FAQ or watch CSE30 Development Tools Tutorial**
(Do NOT wait until Friday to try this. There will be limited or no ETS support on the weekends!)

We've provided you with the starter code at the following link:
https://github.com/cse30-fa21/A1-starter

1. Download to get the `Makefile`, `README`, and empty `checksum.c` with template header.
    a. You can either download the repo locally and SCP the folder, or use `git clone`.
        i. More information in FAQ and video
2. Fill out the fields in the `README` before turning in

## Running Your Program

We've provided you with a `Makefile` so compiling your program should be easy!

Additionally, we've placed the reference solution binary at:
`/home/linux/ieng6/cs30fa21c/public/bin/checksumA1_ref`.
You can directly use `checksumA1_ref` as a command.

**Makefile**: The `Makefile` provided will create a `checksum` executable from the source files provided. Compile it by typing `make` into the terminal. By default, it will run with warnings turned on. To compile without warnings, run `make WARN=` instead. Run `make clean` to remove files generated by `make`.

# How the Program Works

Write a simple C program that reads characters from standard input (`stdin`) using `getchar()` and writes those characters to standard output (`stdout`) using `putchar()`. The program should accumulate a checksum of all characters written and print the base-10

checksum after encountering an EOF (end of file). This checksum should be printed to stdout as well, using `printf()`. When reading from stdin, EOF can be inputted using CTRL+D (control key and D pressed at the same time).

For detailed information about the functions above, you can enter `man [function]` in the terminal.

The checksum is computed in a **`char` variable that is initialized to 255**. As each character is read from the standard input, it is added to the checksum. Any overflow from the checksum variable is ignored (e.g. 255 + 10 = 9). The checksum computation should include all characters up until the EOF. When all the characters have been written, the checksum is then written as a decimal integer. When printing, be sure to follow the checksum with a newline.

As an example, suppose we have a file called `Endermen` as shown:

```
% cat Endermen
All your dirt are belong to us
- Endermen
%
```

After compiling our program, we get an executable called “checksum”. You may recall from your 15L or equivalent course that you can assign stdin to be the contents from a file using the “<” operator.  Here is our program running with the file Enderman as its standard input.

```
% ./checksum < Endermen
All your dirt are belong to us
- Endermen
99
%
```


(this sign is not perfectly accurate since it has 3 lines)

You should write a short `checksum.c` that uses `getchar()` and `putchar()` in an appropriate way to print the characters to the screen and then uses `printf()` in an appropriate way to print the checksum after an EOF is encountered.

For a hint to get started, here’s an example that uses `getchar()` and `putchar()`.

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(void) {
   int c;

   while ((c = getchar()) != EOF) {
       if (islower(c))
           c = toupper(c);
       (void)putchar(c);
   }
   return EXIT_SUCCESS;
}
```

**NOTE: EOF on the Pi-Cluster machines compiles to -1, and `getchar()` returns an `int`. Make sure to assign the value returned from `getchar()` to a variable of type `int`, or you may run into problems.**

# Submission and Grading

## Submitting

1. Submit your files to Gradescope under the assignment titled “A1: Checksum”.
   You will submit the following files:

   ```
   checksum.c
   README.md
   ```

   You can upload multiple files to Gradescope by holding CTRL (⌘ on a Mac) while you are clicking the files. You can also hold SHIFT to select all files between a start point and an endpoint.

   Alternatively, you can place all files in a folder and upload the folder to the assignment. Gradescope will upload all files in the folder. You can also zip all of the files and upload the `.zip` to the assignment. Ensure that the files you submit are not in a nested folder.

2. After submitting, the autograder will run a few tests:
   a. Check that all required files were submitted.
   b. Check that `checksum.c` compiles.
   c. Runs some sanity tests on the resulting `checksum` executable.

## Grading Breakdown [25 pts]

**Make sure to check the autograder output after submitting!** We will be running additional tests after the deadline passes to determine your final grade. Also, throughout this course, make sure to write your own test cases. ***It is bad practice to rely on the minimal public autograder tests.***

The assignment will be graded out of 25 points, each test case equaling 2.5 points, with a total of 10 test cases. (There are 5 public test cases.) Make sure your assignment compiles correctly through the provided `Makefile` on the pi-cluster. **Any assignment that does not compile will receive 0 credit.**