The objective of this assignment is for you to set up the infrastructure you need for subsequent assignments and to write and execute basic MIPS and C programs. Help is available through **piazza**, during **TA office hours** (posted on Canvas syllabus page) and by sending **email** to ece2035-help@ece.gatech.edu. There are also links to resources, tutorials, primers, and installation guides on Canvas (Modules > Introduction).

This assignment has three parts, all due at the same time.

HW1-1: The goal of this part is to use a Linux/Unix environment and become familiar with its facilities. Toward this end, you must acquire/access and run an appropriate Linux/Unix distribution that fits your computing environment. The following are some the options available (see Resources and Installation Guides on Canvas):

- Unix underlying Mac OS X
- Linux Bash shell on Windows 10 using the built-In Windows Subsystem for Linux
- Running native Ubuntu
- VMware on Windows 8

Once you do this, then perform the following steps. Recommended: Dr. Gong Chen created an excellent video on using Linux commands, the command line interpreter (terminal/shell), and gcc which can be found on Canvas > Modules > Introduction > Getting Started Demos.

1. Create and edit an ASCII text file in your favorite text editor that contains the following:
   - Tell one interesting fact about yourself.
   - The method you are using to run Linux/Unix (e.g., one of the options above).
2. Open a Linux command line interpreter (terminal/shell) and change to the directory that contains the text file you created in step 1.

   You can use the Linux command **`cd`** to change your current working directory to the directory in which you saved your text file. For example,

   ```
   gburdell@GTLaptop:~$ cd /mnt/c/Users/gburdell/2035/hw1
   ```

   In this example, the text before and including "`$`" is the command line prompt. The rest of the line is the command. (Note for Windows10: If you are using the WSL/Linux bash shell, the "`/mnt/c/...`" path allows you to access files on your local Windows drive from the bash shell.)

   Type "**`man cd`**" or, more generally,"**`man`** *command_name*" to see the manual page documenting a command.

   You can use the **`ls`** command to list the files in the directory. For example,

   ```
   gburdell@GTLaptop:~$ ls -lt
   ```

   This example uses the -l and -t options to use the "long listing format" and "list in sorted order by modification time," respectively. Type "`man ls`" for information on all the options.

3. Use the **`cat`** command to view the contents of your file. (You can also use the commands **`more`** or **`less`**.) For example, use the **`cat`** command to view the contents of a file like this:
   ```
   gburdell@GTLaptop:~$ cat myfacts.txt
   ```
4. Capture a screenshot which shows your terminal window with your text file contents displayed on it. Submit your screenshot in **jpeg format**.

In order for your solution to be properly received and graded, there are a few requirements.

1. The file must be named **`HW1-1.jpg`**. (It is OK if Canvas renames it slightly by adding an extra version number to it, e.g., HW1-1-2.c or HW1-1-5.c. Canvas does this whenever you upload a newer version of your submission. We will grade the most recent submission you upload.)
2. Your solution must be properly uploaded to the Canvas site (under Assignments > HW1-1) before the scheduled due date.

HW1-2: The goal of this part of the project is to modify a short C program, compile it using the GNU C Compiler `gcc`, and run it. A program shell `HW1-2-shell.c` is provided. You must copy/rename it to `HW1-2.c` and modify it to compute a measure of overlap between two rectangles R1 and R2, called *intersection over union*. It is defined as:

$$IoU = \frac{Area(R1 \cap R2) * 100}{Area(R1) + Area(R2) - Area(R1 \cap R2)}$$

For example, if the rectangles are the same area 30 and their area of overlap is 10, IoU = 20%. If they do not overlap, IoU=0%, and if they completely overlap and are the same size, IoU=100%.

This is frequently used in computer vision to measure how closely aligned two bounding boxes are to each other. For example, comparing the location of a detected object of interest with that of known correct "ground truth" location can be used to evaluate the accuracy of an object detection algorithm. If the IoU is greater than some threshold, the detected object would be counted as a true positive. See Fig. 1.

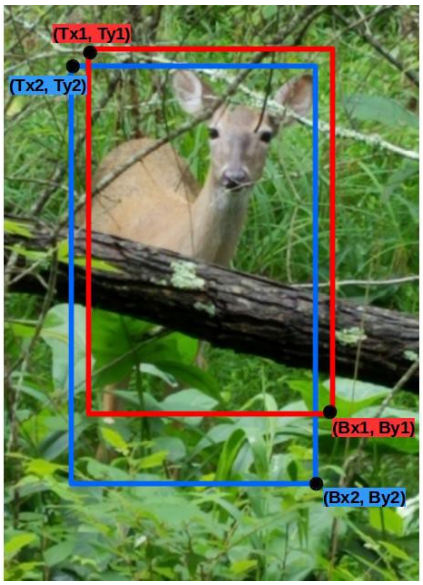

Fig 1: Example image annotated with bounding boxes for ground truth (blue) and detected object (red).

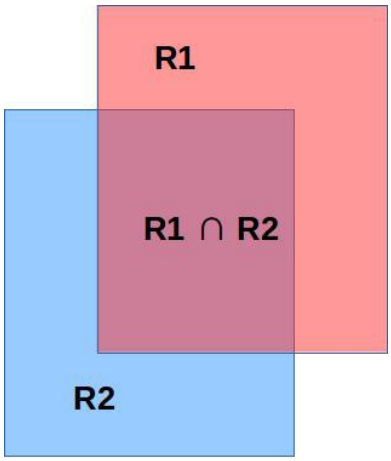

Fig 2: Intersection over Union

Each bounding box is specified by the coordinates of their top left corner (Tx, Ty) and bottom right corner (Bx, By), given as the four elements of an array in this order: {Tx, Ty, Bx, By}. For example,

```
int R1[] = {64, 51, 205, 410};
```

specifies the bounding box with top left corner (64, 51) and bottom right corner (205, 410).

Important details, simplifying assumptions, and hints:

- In images, the origin (0,0) is located at the upper leftmost pixel, x increases to the right and y increases downward. So in our bounding box representation, it will always be true that: Tx < Bx and Ty < By.
- Assume images are 640x480 and bounding boxes fit within these bounds and are always of size at least 1x1.
- For this homework, make the following simplifying assumptions:
    1. the bounding boxes do overlap (assume $Area(R1 \cap R2) > 0$ and IoU >= 1%).
    2. the relative positions of R1 and R2 are restricted to the single case shown in Figure 2: only R1's *bottom left* corner is inside the bounding box R2 while R1's other 3 corners are not, and only R2's *top right* corner is inside R1.
- IoU should be specified as an integer percent (only the whole part of the division), i.e., round down to the nearest whole number between 1 and 100, inclusive.
- Be sure to multiply the area of intersection by 100 *before* dividing it by the area of union. You are doing integer arithmetic, not floating point operations. You'll lose significant digits (precision) if you multiply by 100 after the division.
- Try multiple test cases, but do not change the declaration of the global variables (you should change only their initial values, such as the elements in the arrays to create new test cases).

Compiling and running your code:

You should open a command line interpreter (terminal/shell) to run `gcc`. (For information on gcc, watch Dr. Chen's demo, type `man gcc` for compiler usage, and/or look up GCC online documentation on the internet.)

If you do not have gcc installed, you can use "sudo apt-get install gcc" to install it. See these links for quick tutorials on apt-get and installing packages:

- https://www.howtogeek.com/261449/how-to-install-linux-software-in-windows-10s-ubuntu-bash-shell/
- www.howtogeek.com/63997/how-to-install-programs-in-ubuntu-in-the-command-line/

Note that in the terminal window, you can enter any of the Linux commands (such as `ls`, `cd`, `cp`; for reference see the Linux Command Quick Reference sheet under Canvas > Modules > Introduction > Readings/Resources.

You can copy a file using `cp` or rename a file using `mv` (move a file to a new file). For example:

```
gburdell@GTLaptop:~$ cp HW1-2-shell.c HW1-2.c
```

Using the ASCII text editor of your choice modify the `HW1-2.c` program.

Once you write your program, you can compile and run it on the Linux command line:

```
gburdell@GTLaptop:~$ gcc HW1-2.c -g -Wall -o HW1-2
gburdell@GTLaptop:~$ ./HW1-2
```

You should become familiar with the compiler options specified by these flags.

In order for your solution to be properly received and graded, there are a few requirements.

1. The file must be named **`HW1-2.c`**. (As mentioned above, it is OK if Canvas renames it slightly with an extra version number; we'll grade your most recent submission.)
2. Your name and the date should be included in the header comment.
3. *Do not* `#include` *any additional libraries.*
4. It is especially important not to remove or modify any global variable declarations or print statements since they will be used in the grading process.
5. Your solution must include proper documentation and appropriate indentation.
6. Your solution must be properly uploaded to Canvas before the scheduled due date.

HW1-3: The goal of this part is for you to install MiSaSiM, modify a short assembly program `HW1-3-shell.asm`, simulate, test and debug it in MiSaSiM. The MiSaSiM simulator can be installed according to the instructions at http://lindawills.ece.gatech.edu/misasim/. Copy or rename the shell program to `HW1-3.asm` and modify it to compute the *intersection over union* of two rectangles which are allocated and initialized in the .data section of the program.

These use the same bounding box format as in part HW1-2. Store the result (an integer between 0 and 100, inclusive) in the memory location labeled `IoU`.

The same details, assumptions, and hints given in HW1-2 apply in HW1-3. Two additional constraints *for this assignment only are*:

- Do not write values to registers $0, $29, $30, or $31.
- Do not use helper functions or function calls (JAL instruction).

Be sure to try multiple test cases. To create new test cases, you may change the initial values of any global data defined in the .data section of the program, but for HW1, do not remove or change any labels provided for the data.

In order for your solution to be properly received and graded, there are a few requirements.

1. The file must be named **`HW1-3.asm`**. (As mentioned above, it is OK if Canvas renames it slightly with an extra version number; we'll grade your most recent submission.)
2. Your name and the date should be included in the beginning of the file.
3. Your program must store its result at the memory location labeled `IoU` when it returns. This answer is used to check the correctness of your code.
4. Your program must return to the operating system via the **`jr`** instruction.
5. *Programs that include infinite loops or produce simulator warnings or errors will receive zero credit*.
6. Your solution must include proper documentation.
7. Your solution must be properly uploaded to Canvas before the scheduled due date**.**

Honor Policy: In all programming assignments, you should design, implement, and test your own code. **Any submitted assignment containing non-shell code that is not fully created and debugged by the student constitutes academic misconduct.** The only exception to this is that you may use code provided in tutorial-0.zip or in the code examples on the ECE2035 Canvas site (under Modules). **All code (MIPS and C) must be documented for full credit.**