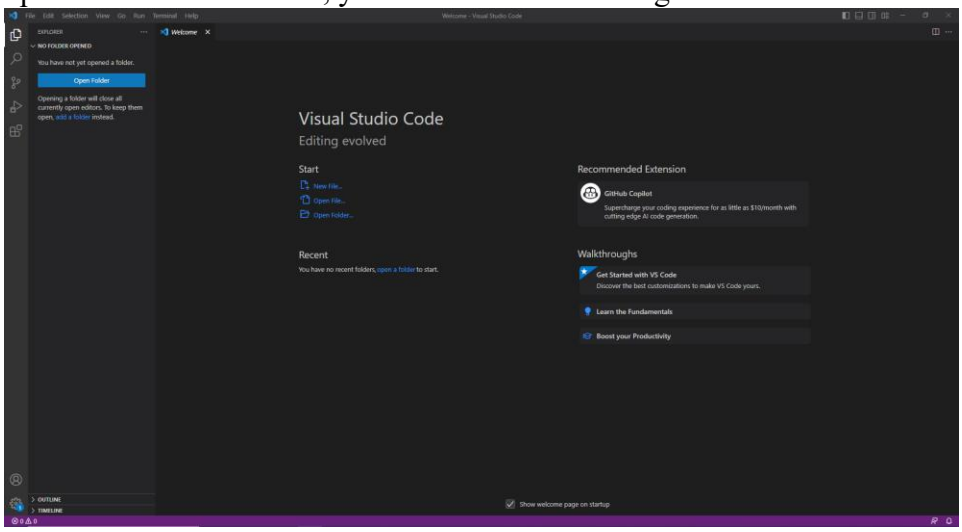
The objective of this assignment is for you to set up the infrastructure you need for subsequent assignments and to write and execute basic MIPS and C programs. Help is available through **Ed Discussion**, during **TA office hours** (posted on Canvas syllabus page) and by sending **email** to ece2035-help@ece.gatech.edu. 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.

In working this assignment, it is helpful to first complete the MIPS Tutorial which may be found under Assignments on Canvas.

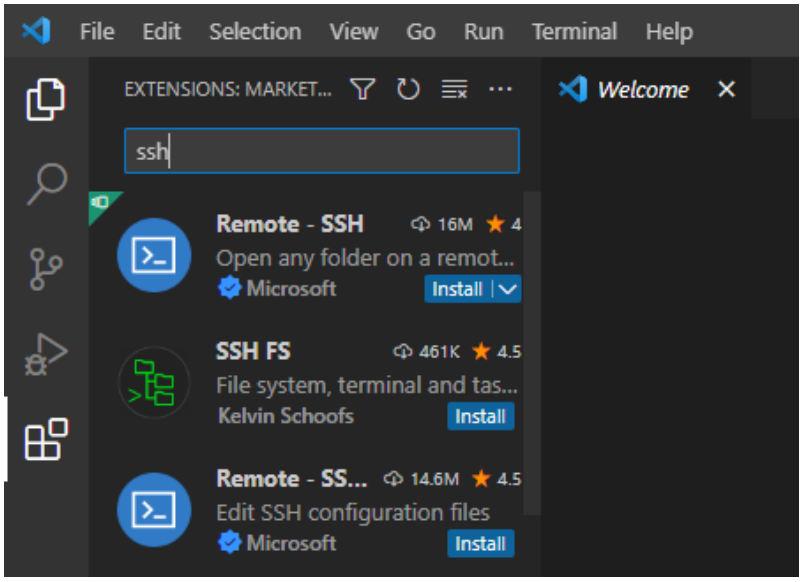
HW1-1: The goal of this part is to use a Linux/Unix environment and become familiar with its facilities. Toward this end, you must access the ECE Linux lab machines. This document will walk through how you should connect to the Linux lab machines. The following steps work on both Windows and macOS. There is also a video demonstrating the process available under Canvas Modules > Introduction > **Installing VSCode and Accessing Linux Lab Machines**.

The first time you perform these steps will take a few minutes. Please be patient and don't worry. The process of connecting and running your code once you have everything set up is faster.

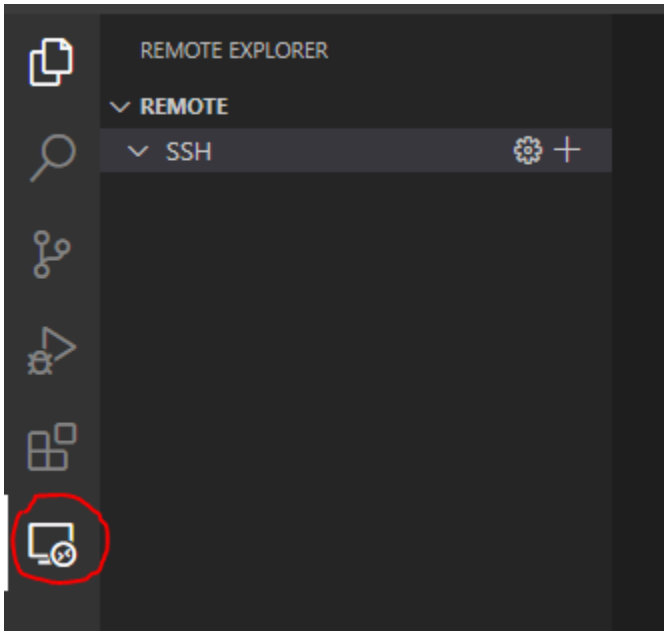
1. **Install VSCode:** Go to download Visual Studio Code, select the appropriate version for your device and install it (the default installation options are fine). Once installed, open Visual Studio Code, you should see something similar to this:



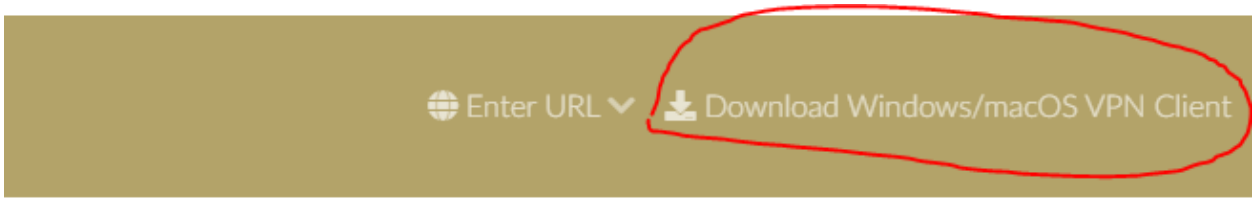
2. **Install ssh extension** for Visual Studio Code: Click the 4 blocks icon on the left hand side to open up the extensions marketplace. Search for "ssh" and select the first option shown in the image:

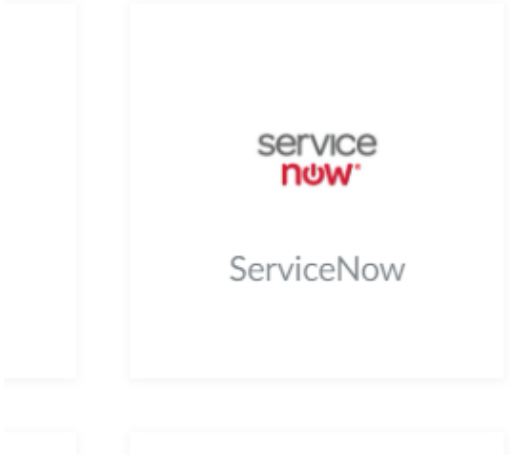


Install it and you should see this new icon below the 4 blocks icon:

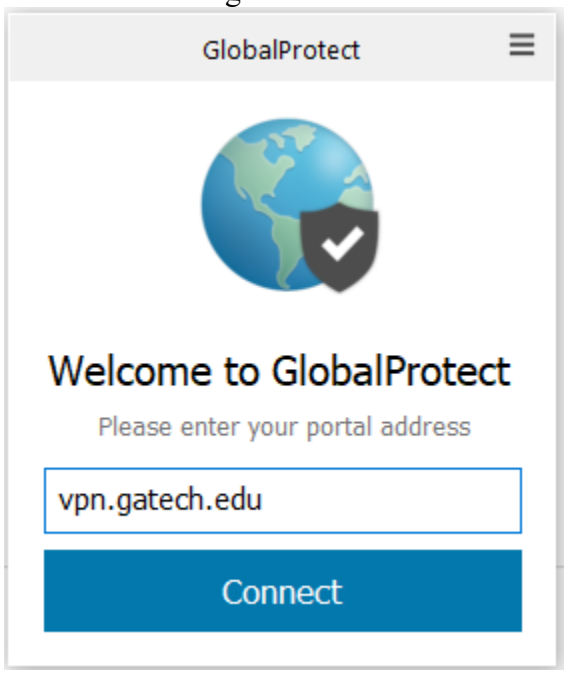


3. **VPN client**. If you have not installed the VPN client to connect to the Georgia Tech VPN you will do so in this step. First go to the Georgia Tech VPN website and log in. On the top banner, you will see a link to download the Windows/macOS VPN client. Follow the link and install the appropriate version for your device.



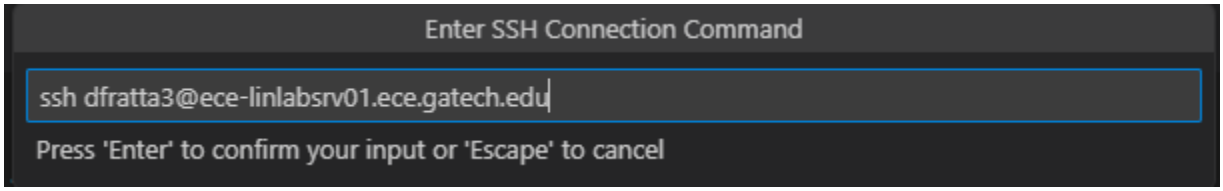


Now open and run GlobalProtect, which is the VPN client. It will be asking for your portal address. Enter vpn.gatech.edu as your portal address and hit connect as demonstrated in the image:

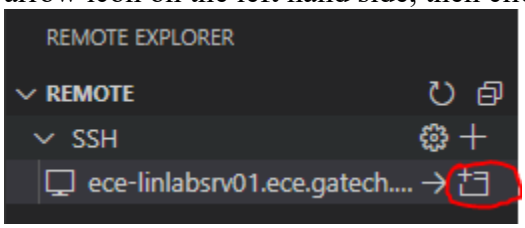


Once you press connect, it will ask you to do an MFA login with your Georgia Tech credentials and it will say when you are connected in the client. Once connected we return to Visual Studio Code.

4. **Use ssh to connect within VSCode**: Click the plus sign in the SSH row under the remote explorer icon (the one highlighted in step 2). You should then see a small pop up to connect. In the pop up you should type
   `ssh <GT_USERNAME>@ece-linlabsrv01.ece.gatech.edu`
   It will ask you to save the configuration, select the .ssh/config option. (On Windows, this looks like "`C:\Users\gburdell\.ssh\config`".)

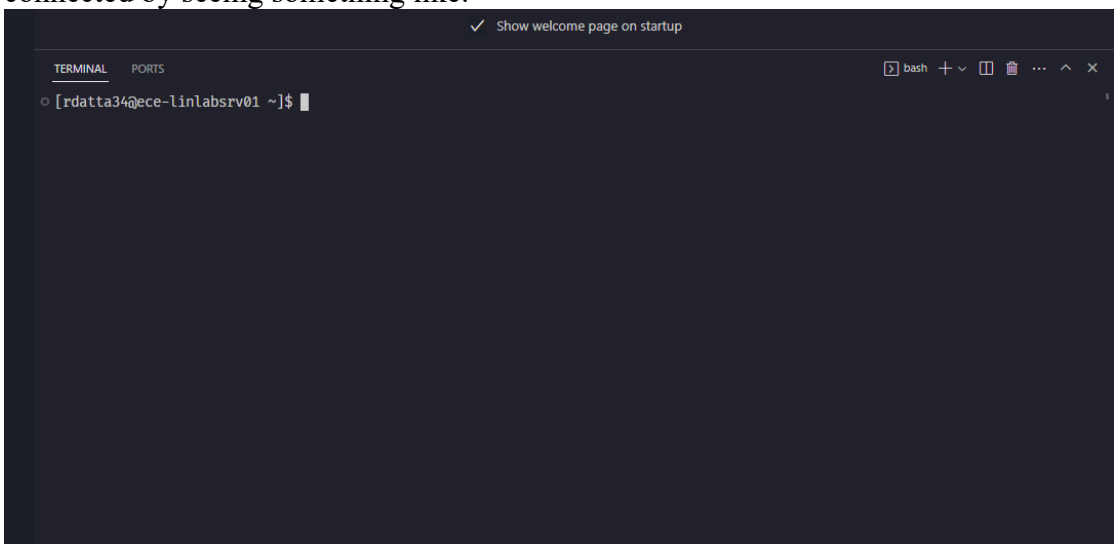


   Once you have done that, refresh the Remote Explorer tab by clicking the circular arrow icon on the left hand side, then click the connect button



   You can only connect if you are on the Georgia Tech VPN, so make sure you are connected to the VPN before attempting to connect to the servers. It will then ask you some questions about the connections. For the operating system, select Linux, and for the Certificate you should accept it. Then it will ask you for your Georgia Tech password. Once you correctly enter your password, you should be connected in a new window of Visual Studio Code!

5. **Open a terminal** by clicking on Terminal in the top menu bar – or on the three dots to allow you to select Terminal - and then select New Terminal. You can verify you are connected by seeing something like:



   It is important that you see your username@ece-linlabsrv01 in the terminal. *At this point, you can close the first VSCode window you opened before connecting remotely.*

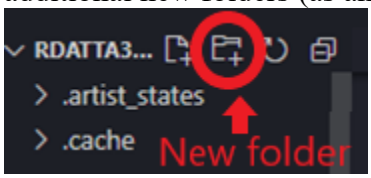
If you need help getting these steps completed, please drop by office hours.

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 > Demo of essential Linux**.

6. **Create/open folders in Terminal/Explorer**: From the terminal, create a directory (aka folder) that will hold your files. For example,

   ```
   [gburdell@ece-linlabsrv01 ~]$ mkdir 2035
   ```

   You can also view the directory tree in the **Explorer pane** on the left by clicking on View in the top menu bar and selecting Explorer (or by clicking on the Explorer icon at the top of the left vertical icon pane). From there, you can click on Open Folder, which lets you browse to and select a folder. In the Explorer pane, you can create additional new folders (as an alternative to using mkdir in the terminal pane).



7. **Transfer files to remote server**: Download **HW1-1.txt** from the Canvas assignment page and transfer it to your directory on the Linux Lab server – you can drag and drop it into the Explorer pane in VSCode. Open your file in the VSCode editor and **edit the file to fill in the blank with one interesting fact about yourself and save it.**

8. **Navigate to a folder in the Terminal:** Using the Visual Studio Code terminal, change to the directory that contains your edited HW1-1.txt. (If Terminal is not currently visible, you can click on View in the main top menu bar and select Terminal.)

   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@ece-linlabsrv01 ~]$ cd 2035/hw1
   ```

   In this example, the text before and including "`$`" is the command line prompt. The rest of the line is the command. This example assumes that hw1 is a folder previously created under the 2035 folder.

   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@ece-linlabsrv01 hw1]$ 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.

9. **View contents of text file in terminal.** 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@ece-linlabsrv01 hw1]$ cat HW1-1.txt
   ```

10. **Capture a screenshot** which shows your VSCode window with the editor and the terminal both showing your text file contents displayed on it. Submit your screenshot in **jpeg format** in a file named **HW1-1.jpg**. There are multiple ways of taking a screenshot. On MacOS: cmd-shift-5. On Windows: win+shift+s or Alt+PrtScr, and then pasting from clipboard to a file in Paint or Paint3D.

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. It must compile and run without errors on the ECE Linux Lab machines – for consistency, all student code submissions will be graded using the version of gcc running on these machines. To prepare for this part, please be sure to install the C/C++ extensions and the ssh keys as described in the video **Installing VSCode and Accessing Linux Lab Machines** (starting at min 10:35) under Canvas > Modules > Introduction.

A program shell `HW1-2-shell.c` is provided. You must copy/rename it to `HW1-2.c` and modify it to determine whether the square of a given integer `x` can be calculated by the following "trick" method.

It is suggested that the square of any number `x` between 9 and 32,768 can be calculated by

1. Separating `x` into two numbers Xhi (everything except the last decimal digit) and Xlo (the last decimal digit). For example, for `x` =54, Xhi is 5 and Xlo is 4. And for `x`=1492, Xhi is 149 and Xlo is 2.
2. Computing Xhi+Xhi^2 (i.e., Xhi + the square of Xhi). **For `x` =54**, where Xhi is 5, this is 5+5^2, or **30**.
3. Computing the square of Xlo, padding it as necessary to make it two digits. For `x`=54, where Xlo is 4 this is **16**, but if Xlo had been 2, this would result in 04, NOT 4.
4. Concatenating the result of step 2 and step 3, with step 2 on the left, step 3 on the right. **For `x` =54**, the final result is **3016**.

Let's call the value computed by this method `A`.

For any given 9 < `x` < 32,768, your program should compute the square of `x` and store it in variable `SQ`, and it should compute `A` using the method described above. If `SQ` and `A` are equal, it should set variable `Result` to 1, otherwise it should set `Result` to 0.

In addition, once you've tried out several test values for `x`, fill in the answer to the question in the comments at the top of the C file. (What is true of all integers `X` for which `SQ == A`, but is not true for integers where `SQ != A`?)

Use the print statement provided in the shell code. Do not modify it. If you add any additional print statements while debugging your code, be sure to comment them out before handing in your code so that the autograder grades the intended output.

Try multiple test cases, but do not change the declaration of the global variables (you should change only the initial value of `x` to create new test cases).

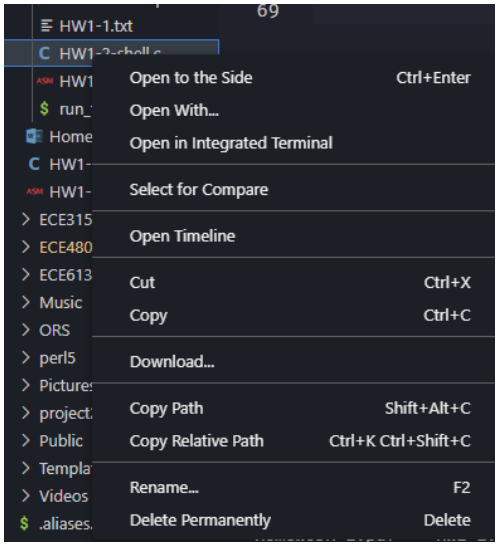
Compiling and running your code:

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.)

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

```
[gburdell@ece-linlabsrv01 hw1]$ cp HW1-2-shell.c HW1-2.c
```

You can also drag and drop local files to the Linux lab servers from your local computer in the file explorer on Visual Studio Code and Rename a file using the VSCode gui:



Using the VSCode editor or any 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@ece-linlabsrv01 hw1]$ gcc HW1-2.c -g -Wall -o HW1-2
[gburdell@ece-linlabsrv01 hw1]$ ./HW1-2
```

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

Testing and debugging your code:

Be sure to try multiple test cases. One test case is provided in the global variables in the shell code. You may change the *initial values* of any global variables to create new test cases, but do not remove or change any global variable type declarations.

We have also provided a short **"smoke test" script (run_tests.sh)** with the HW1-2 Assignment on Canvas. Smoke testing is a preliminary stage of development that checks for glaring errors such as the code does not compile or the answer produced is printed out incorrectly (e.g., it looks for extra/missing characters in the print statement or for extra debugging print statements that

were left in). *Please place* **run_tests.sh** *in the same directory where you put your HW1-2.c code.*

You must run the smoke test on your code on the Linux Lab servers before submitting your code and fix any errors detected.

```
[gburdell@ece-linlabsrv01 hw1]$ ./run_tests.sh
```

If this gives the error "bash: ./run_tests.sh: Permission denied" be sure to change the permissions on the bash script file (run "man chmod" at the command line for more information):

```
[gburdell@ece-linlabsrv01 hw1]$ chmod u+x run_tests.sh
```

The **run_tests.sh** script will point out errors that you should fix before submitting your code. The output will also be saved to a text file **log.out** in the same directory.

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 inserted in the header comment.
3. *Do not* `#include` *any additional libraries.*
4. Answer the question in the comments: What is true of all integers x for which SQ == A, but is not true for integers where SQ != A?
5. 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.
6. Before submitting your code, run it through the smoke tests to ensure that your code is providing the answer in the proper format without any extraneous print statements. ***If your submitted code has a bug that the smoke test would detect, you will receive a 0.***
7. Your solution must include proper documentation (comments) and appropriate indentation.
8. 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 given in the Canvas module > Introduction > MISASIM Installation Guide. Copy or rename the shell program to `HW1-3.asm` and modify it to compute the same information as in HW1-2. In this part, the input `x` is given in the memory location labeled `xAddr` (as defined in the `.data` section of `HW1-3-shell.asm`.) Instead of printing the results, store the values of `A`, `SQ`, and `Result` in the locations allocated for them in the `.data` section.

Two additional constraints apply *for this assignment only*:

- 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 (or their order).

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 inserted at the beginning of the file.
3. Your program must store `SQ`, `A`, and `Result` at the memory locations labeled `SQAddr`, `AAddr`, and `ResultAddr`, respectively. This answer is used to check the correctness of your code. Do not change the order of these labels in the `.data` section.
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.