24783 Advanced Engineering Computation: Problem Set 5

(*) In the following instruction (and in all of the course materials), substitute your Andrew ID for where you see *yourAndrewId*.

START EARLY!

1 Check Out or Update Base Code and Libraries

Please make sure you have up-to-date libraries and course files before starting an assignment.

If you have not done working-directory set up as described in the first assignment (like in case you need to work from a different computer), please see Problem Set 1 and set up the working directory.

I assume you created the working directory called *24783* under you home directory and you checked out your Git repository in there.

Home directory is typically *C:\Users\username* in Windows, */Users/username* in macOS, and */home/username* in Linux, where *username* is the user name in your local computer.

First, open command-line (Developer PowerShell or Terminal), and move to your working directory by typing:

```
cd ~/24783
```

You need to check out (or clone) Git repositories once. If you have not checked out yet, do the following:

```
git clone https://yourAndrewId@ramennoodle.me.cmu.edu/Bonobo.Git.Server/course_files.git
git clone https://yourAndrewId@ramennoodle.me.cmu.edu/Bonobo.Git.Server/yourAndrewId.git
```

You need to replace "yourAndrewId" with your Andrew ID. You'll be asked to type in credentials.

Also we are going to use two additional repositories:

```
git clone https://github.com/captainys/MMLPlayer.git
git clone https://github.com/captainys/public.git
```

If you are successful, you should have the following directory structure under your home directory.

```
Your User Directory
├── (Other files and directories)
└── 24783
    └── course_files
```

```
└── yourAndrewID
```

If you already have checked out these repositories (most likely you did for Problem Set 1), you need to update (or git pull) in those repositories. By change directory to the location where you checked out repositries and then type:

```
git pull
```

To update all four repositories, you can type the following commands in a sequence:

```
cd ~/24783/course_files
git pull
cd ~/24783/yourAndrewID
git pull
cd ~/24783/public
git pull
cd ~/24783/MMLPlayer
git pull
```

2 Copy Base Code and Add to Git's Control

Like ps1 through ps5, you can start from the base code. Copy ps5 directory to your working directory. I assume your working directory is:

```
~/24783
```

and, your copy is:

```
~/24783/ps5
```

After copying, your directory structure should look like:

```
Your User Directory
├── (Other files and directories)
└── 24783
    ├── course_files
    ├── public
    ├── MMLPlayer
    └── yourAndrewID
        └── ps5
            ├── ps5.pdf
            ├── ps5_1
            ├── ps5_2
            ├── hash
            └── bitmap
```

It is important that public, MMLPlayer, and yourAndrewID directories are at the same level, and ps5 directory is directly under yourAndrewID directory. Otherwise your program cannot be built in the grading environment.

3 Make CMake Projects

Write cmake scripts. You need to write one top-level script, and one each for a sub-directory (except png sub-directory). Therefore, you will write five CMakeLists.txt files in total.

The top-level CMakeLists.txt must add sub-directories public, MMLPlayer/ym2612, and MMLPlayer/mmlplayer.

Make sure to use target names, ps5_1, ps5_2, hash, and bitmap (same as the directory name, all small cases.)

Library target "bitmap" must link to ysbitmap library included in the public repository because it uses my PNG decoder.

When you use add_subdirectory, use relative path. Absolute path like C:/Users/soji/24783/soji does not exist in the grading environment. Also use slash instead of backslash. Windows can accept both backslash and slash. Other platforms only accepts slash. I want you to learn cross-platform programming.

4 Bitmap Comparison (20 points)

Write a program that:

1. Reads two .PNG bitmaps (let's call them bitmaps A and B) specified by the first and second command arguments (argv[1] and argv[2]) .
2. Make another bitmap (let's call it bitmap C) that is large enough to cover both two bitmap.
3. Clear the new bitmap with R=G=B=A=0.
4. For each pixel coordinate (X,Y), if any of the RGB values of bitmap A is different from bitmap B by more than 60 (in 0-255 scale), put R=255,G=0,B=0,A=255 to the pixel at (X,Y) of bitmap C. If (X,Y) is outside of bitmapA or bitmapB, consider it as different.
5. Open a window that bitmap C fits, and render bitmaps on the window. The program must have three rendering modes:
   (a) Mode 0: Draw Bitmap A.
   (b) Mode 1: Draw Bitmap B.
   (c) Mode 2: Draw Bitmap C.
6. The program starts in Mode 0. When the user presses 0 key (FSKE_0), 1 key (FSKEY_1), or 2 key (FSKEY_2), change mode to Mode 0, 1, or 2, respectively.

The goal is to find three differences in the pair of pictures. However, due to picture compression, the colors are not exactly same. It may be easier to find the differences by flipping between two pictures between modes 0 and 1. Obvious differences can be highlighted in mode 2.

If the user does not specify two arguments, print "Usage: ps5_1 pictureA.png pictureB.png" and terminate.

If one or both of the two PNG images could not be loaded, print "Cannot open input file." and terminate.

You can test your program with sample bitmaps in the ps5_1 sub-directory. Bitmaps are from Yomiuri Newspaper Online.

The program does NOT have to follow the event-driven style. You can do event-driven or polling-based style.

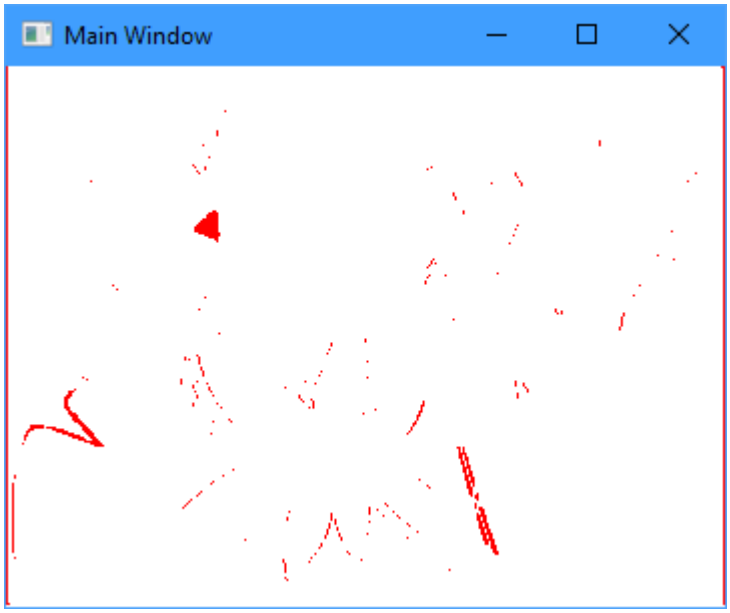
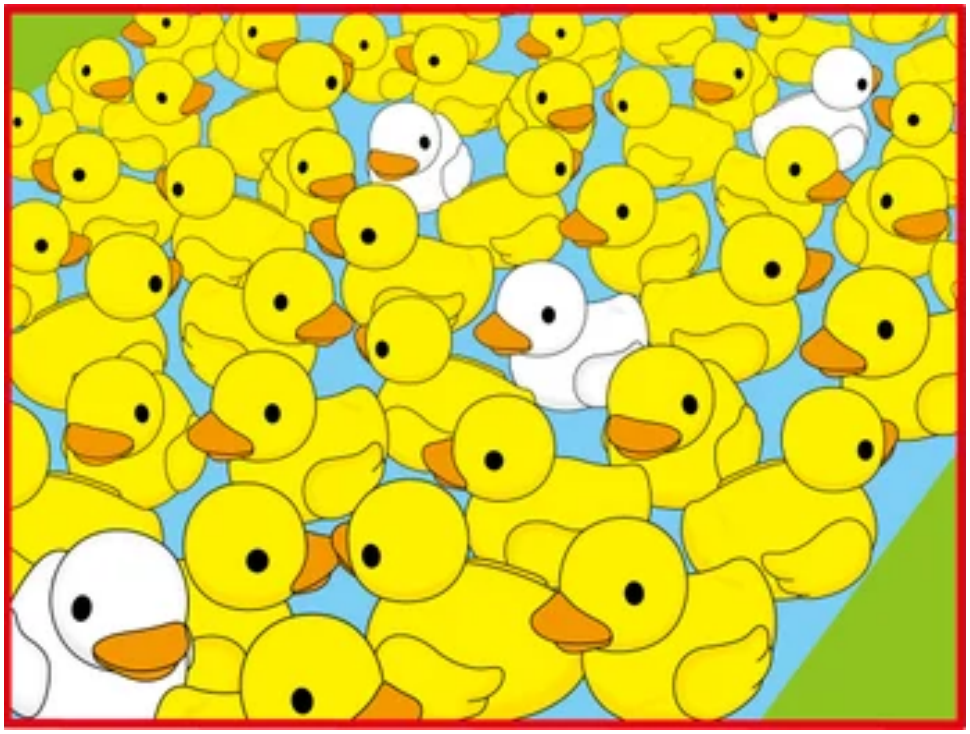


Fig. 1: Mode 2: Highlighting differences between 20220921_leaves_a.png and 20220921_leaves_b.png



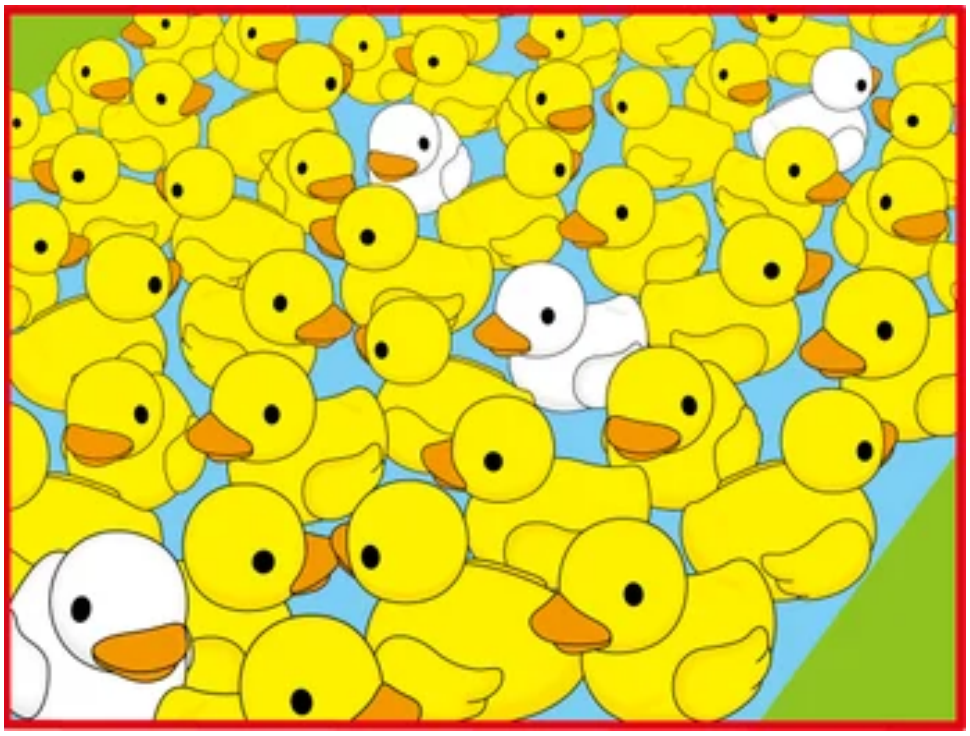


Fig. 2: Can you find three differences?

5 Hydlide Map Browser (80 points)

5.1 Complete CutOut member function

Finish CutOut member function in the SimpleBitmapTemplate class in simplebitmaptemplate.h. The purpose of this function is to copy a sub-region from the *this* bitmap. (I mean this-pointer by *this*.) The function takes a reference to the destination bitmap, top-left corner in the *this* bitmap, and the size of the sub-region to be cut out to the destination. The destination bitmap should be re-sized to the size of the sub-region. If the sub-region extends out of the *this* bitmap, r,g,b,a values (or all the components of the pixel) of the outside of *this* bitmap should be made zero.

To make it easier for you, you don't have to make this function safe for self-cutout (the destination is same as the original.)

5.2 Complete AutoResize Function in the HashTable Class

AutoResize function in the HashTable class is left empty. Fill the function. Resize function is already filled, and you can use the function.

AutoResize is called at the end of insert and erase. The idea is to keep the hash size comparable to the number of elements in the table, which means you need to keep track of the number of elements in the table. Add appropriate member variables and lines in other places in the class.

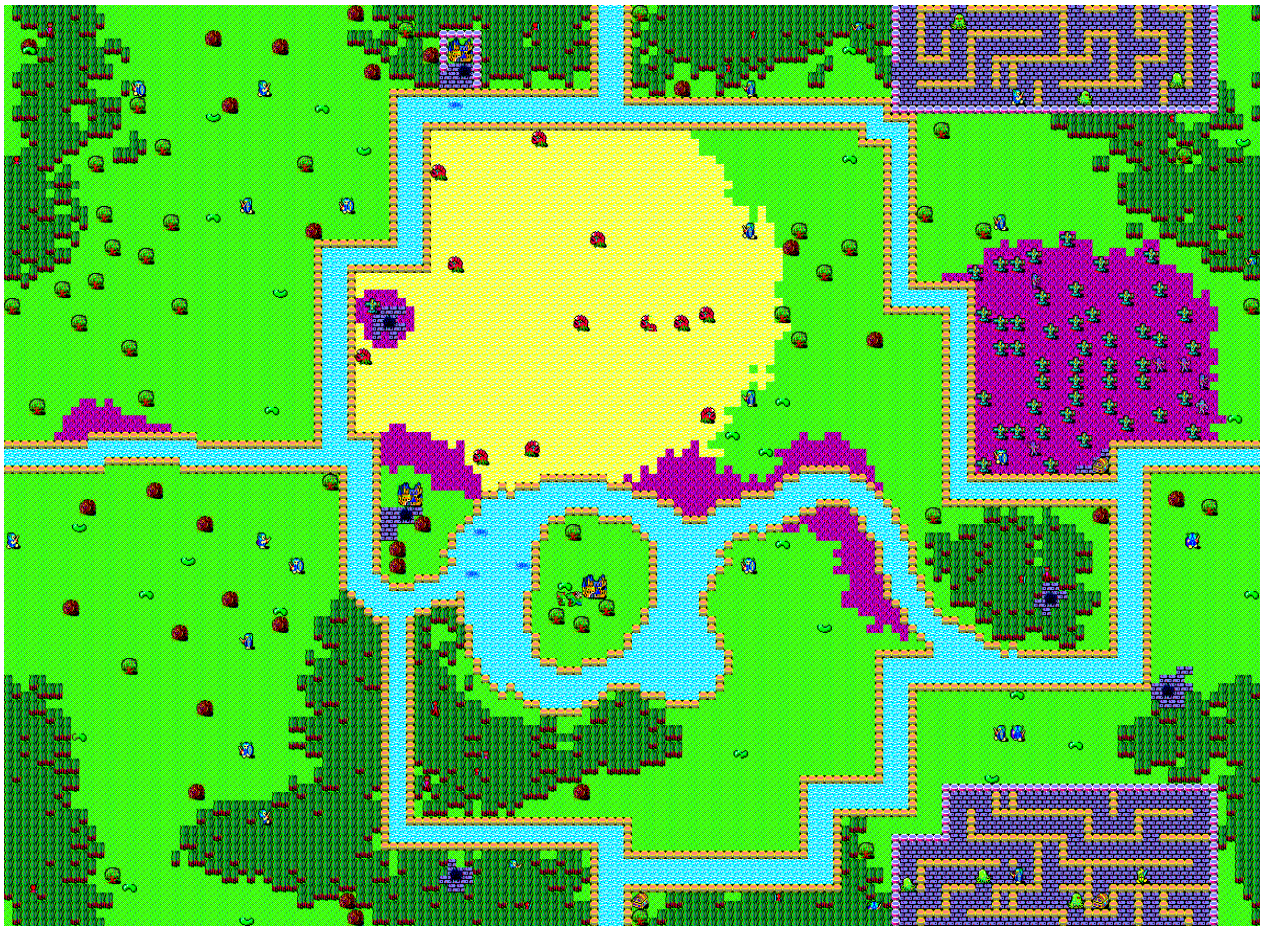


Fig. 3: Field Map from Hydlide (c) T&E Software 1984

5.3 Hash Table of Bitmaps

Fig. 3 is from a retro PC game called Hydlide by T&E Software released in 1984. All the characters were drawn using only 8 colors without support from sprite-rendering hardware. The quality of the pictures was superior than any other games in 1984.

The goal of this assignment is to extract a structure just like you saw in Problem Set 3 base code. In Problem Set 3, the sprite patterns (tiles) were defined in tiles.cpp/tiles.h and the map in map.cpp/map.h. In this Problem Set, you are given a PNG bitmap of entire field map, and you use bitmaps and hash tables to make tiles and map of tile IDs.

Your program in ps5_2 directory must take a PNG file name as input (first argument), and read it into SimpleBitmap data structure. The base code gives an error if the first argument is not given. Even if the first argument is given, if RGBABitmap class failed LoadPng, like due to misspelling of the file, your program needs to print a message and close BEFORE opening the window.

Then, your program makes the following three data structures:

- a hash table that maps a bitmap to an unsigned integer,
- a hash table that maps an unsigned integer to a bitmap,
- a bitmap of unsigned integers and 1 component per pixel (tile map).

Your program needs to cut out 16x16 pixel bitmap (tile) from top-left to bottom-right. If the tile is not already in the hash table, assign an ID to the tile, and insert tile-to-ID and ID-to-tile maps in the two hash tables.

To do this step, complete (1) a hash function for SimpleBitmap in main.cpp, and (2) a hash function for an unsigned integer in main.cpp.

The size of the tile map (bitmap of unsigned integer and 1 component per pixel) needs to be the size of the field-map bitmap divided by 16. And, each pixel of the tile map should store an ID for the corresponding tile.

I think the easiest way to fill the bitmap is to fill it while building the two hash tables. But, you can do it in a separate loop if it is easier for you.

Once IDs are assigned to the bitmaps, open a 800x640-pixel window, and render tiles stored in one of the bitmaps. Either one is fine. Your choice. If you do it correctly, you will see a screenshot like shown in Fig. 4.

Once the user presses the ESC key, show a map browser just like Problem Set 3 solution. Render the map using the tile map and ID-to-tile hash table. When the user presses an arrow key, the map should scroll 4 tiles at a time. If you successfully implement this step, your program should look like, Fig. 5

For a hash function of SimpleBitmap, what you essentially need to do is to calculate an integer value (do not confuse with an ID number) from an array of unsigned chars. There is no unique way of calculating a hash-code from a sequence of unsigned integers. You can come up with your own, or you may try the method described in:

```
http://stackoverflow.com/questions/11128078/android-compute-hash-of-a-bitmap
```

One thing I don't like about this approach is it will for sure let the hash code overflow during the calculation. What you get from overflow is undefined in C/C++ specification.

I rather would go with multiplying different prime number for different location in the array. For example, you may want to multiply:

- 2 to the (5*n)th bytes,
- 3 to the (5*n+1)th bytes,
- 5 to the (5*n+2)th bytes,
- 7 to the (5*n+3)th bytes,
- 11 to the (5*n+4)th bytes,

and add them up.

5.4 5% bonus points for map editor

The bitmap of the field map was made by stitching the screenshots together. Therefore, you see monsters and player characters here and there. I want to clean the map somehow by erasing those monsters and player characters.

One way of doing it is to select a tile with a character and replace with a tile without a character.

In the map-browser mode, let the user select a tile by mouse left-button. And then move the mouse cursor and press R key to replace the selected tile with the tile that the mouse cursor is pointing.

The selected tile must be highlighted by a 2-pixel wide border like shown in Fig. 5

Fig. 4: ps5_2 Output Example (Order may be different depending on how you define your hash function)

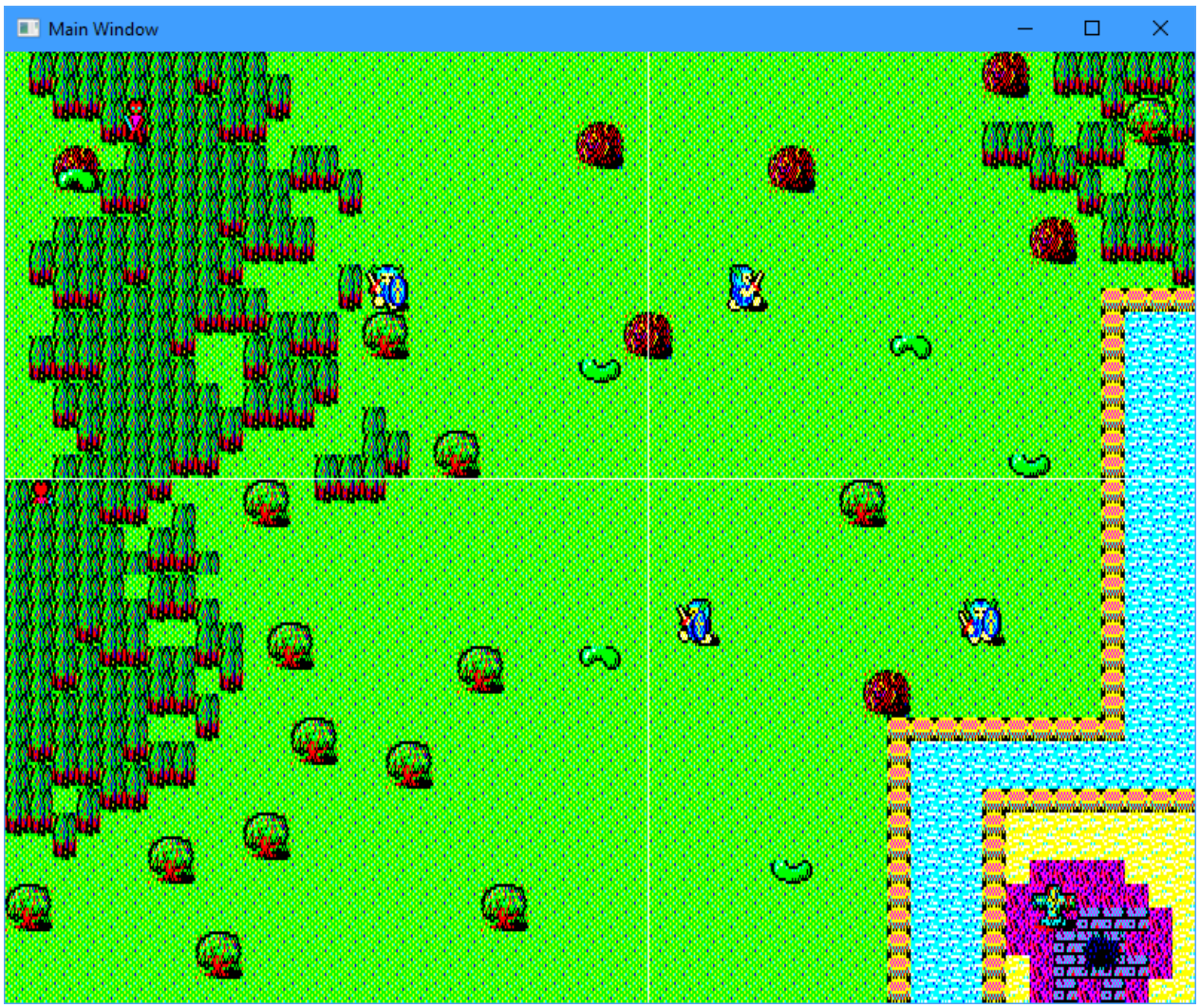


Fig. 5: ps5_2 Output Example (Map browser mode)

Fig. 6: ps5_2 5% extra credit: Selected tile highlighted by a 2-pixel wide border

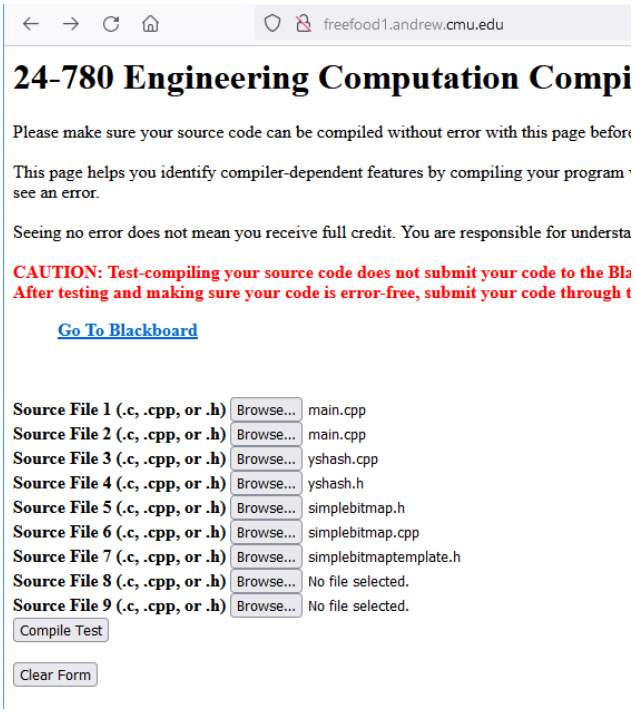


Fig. 7: Test on the Compiler SErver

6 Test Your Code on the Compiler Server and Submit the Screenshot

Test your source files (.cpp and .h files) on the compiler server. Some assignment may not require .h files. You do not have to test files that you don't make modifications. The files you need to test are the ones you write or modify.

We have four compiler servers:

- http://freefood1.lan.local.cmu.edu
- http://freefood2.lan.local.cmu.edu
- http://freefood3.lan.local.cmu.edu
- http://freefood4.lan.local.cmu.edu

Make sure you don't see red lines when you select your files and hit "Compile Test" button on the server.

We have multiple servers to make it less likely that all of them need to shut down for maintenance. If do not have to test on all of the servers. You need to make sure that your code passes on one of the servers.

You can test ps5_1 and ps5_2 together. Select files as shown in Fig. 7 and click "Compile Test".

7 Submit

Lastly, you need to submit using git. What you need to do are two things: (1) add files to git's control, and then (2) send to the git server.

7.1 Add Files to git's control

In this case, you want to add all the files under ps1 subdirectory. To do so, type:

```
git add ~/24783/yourAndrewID/ps5
```

This command will add ps5 directory and all files under the subdirectories.

7.2 Send to the Git Server

In Git, sending files to the server is a two-step process. The first step is local commit. You can do it by:

```
git commit -m "Problem Set 4 solution"
```

The message can be anything, but it is recommended to type something meaningful, at least you can see what changes you made to your repository.

Local commit is just local. Git server does not know about any local commit unless the commit is sent (or pushed) to the server. To do so, type:

```
git push
```

Make sure to do it in the CMU network. If you are working from home (probably most likely), use VPN to connect to the CMU network.

You can re-submit (commit and push) your solution as many times as you want with no penalty before the submission due.

8 Verification

Clone your repository to a different location and make sure that all of your files have been sent to the Git server.

You can do the following:

```
cd ~
mkdir 24783Verify
cd 24783Verify
git clone https://yourAndrewID@ramennoodle.me.cmu.edu/Bonobo.Git.Server/yourAndrewId.git
```

Once you made sure all the files have been submitted, you can delete files and directories under 24783Verify directory.

Good Luck!