ME 459 Assignment 6

Jackson Fox

Due 10/28/2021

Problem 3

a)

A merge conflict occurs when there is a difference in two items being merged together, usually this is the combination of two branches. If I am committing a branch of code that I have been working on back to the main branch, and git finds a piece of data that I changed that has already been modified in the main branch compared to when I created the branch from main, then a merge conflict occurs as my branch cannot be committed without completely overwriting whatever other changes have occurred to the data in question since my branch creation.

b)

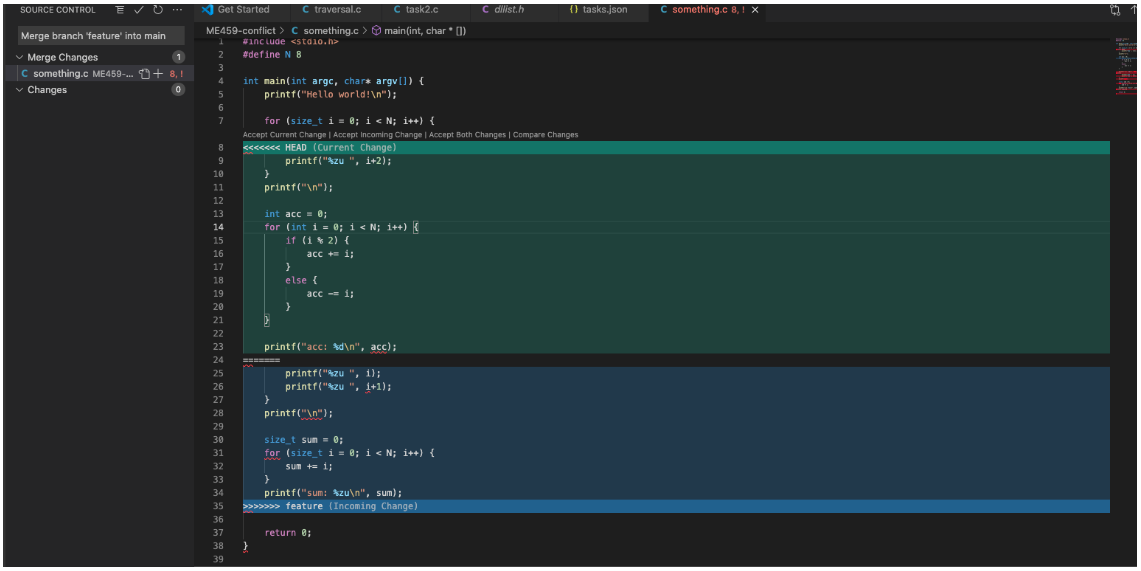
i.)

Something.c has three main output sections, each containing conflicts between the feature and main branches. The program starts with printing ”Hello world!” with the w lowercase in the main and a lower and capital version in the feature. The program then lists either the numbers 2-9 on a line in main or 0 - 8, repeating each number from 1-7, in the feature. The program then returns either an accumulation value about the amount of even numbers from 1 to 7 in main or the value of 7!, both calculated using for loops.

ii.)

Though there was no conflict on the hello world lines, the instructions said to keep the line from main, which included the lower case ”world”, so I deleted the upper case line from the feature version being merged in.

The remaining differences I reconciled using the built in conflict resolver in VS Code. I was able to select if I wanted to include the main, feature or both versions of the changes in the final file, which I selected based on the instructions. See below for screenshot.



SOURCE CONTROL
Merge branch 'feature' into main
Merge Changes 1
something.c ME459-... 8, !
Changes 0
Get Started
traversal.c
task2.c
dllist.h
tasks.json
something.c 8, !
ME459-conflict > something.c > main(int, char * [])
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
#include <stdio.h>
#define N 8

int main(int argc, char* argv[]) {
    printf("Hello world!\n");

    for (size_t i = 0; i < N; i++) {
<<<<<<< HEAD (Current Change)
        printf("%zu ", i+2);
    }
    printf("\n");

    int acc = 0;
    for (int i = 0; i < N; i++) {
        if (i % 2) {
            acc += i;
        }
        else {
            acc -= i;
        }
    }

    printf("acc: %d\n", acc);
=======
        printf("%zu ", i);
        printf("%zu ", i+1);
    }
    printf("\n");

    size_t sum = 0;
    for (size_t i = 0; i < N; i++) {
        sum += i;
    }
    printf("sum: %zu\n", sum);
>>>>>>> feature (Incoming Change)

    return 0;
}
```

Figure 1: VS Code window showing the differences between the feature and main branch versions of the file

After this point, I deleted the main version of the counting method, and then included both changes to get the accumulation and sum portions of the code. I had to do some slight rearranging to make sure that everything was properly ordered in the file when finally saved.

Problem 4

a)

i.)

#29

ii.)

Passed

iii.)

2:59

b)

i.)

Test

ii.)

Yes

iii.)

It tested task 2 with 4, 8, 16 and 25 with the inputs from the command line

iv.)

1 second

Using CI enables you to see quickly if changes that have been committed will build and/or test successfully, without requiring you to pull a full copy of the newly merged repo and attempt to run the program yourself. In the context of working with teams, there is the added advantage of knowing immediately after commits which code/who submitted the code that is causing an issue in the build or successful test of the program/project. This could save lots of time digging through code to understand when an issue arose or whose code caused the problem.