In Class Group Project, Role B

Summary

For this group project, you will be working on two small projects at the same time.

You will create a repository for your project and begin working on it.
You will also share this repository with someone from an opposite Role (A or B), and they will share their project with you.

Once you're done with your part of your project, commit and push your changes. The Opposite role will also do this.
After you push your changes, clone the opposite Role's repository, and begin your changes to their project. They will also pull your changes and update your project.

Once you're done with your changes to the opposite project, commit and push your changes.
Then, in your own project, pull and update the changes from the opposite person.

This will demonstrate working with other people on a project using source control.

Step 1. Create and Share a Repository

In BitBucket, create a new repository. Name it "**Shop**".

Create a folder on your harddrive, and use the **clone** command BitBucket tells you to pull your repo to the harddrive.

Find a **Role A** person and get their BitBucket email address. On your repository webpage, click "Invite" and enter their Email Address. Set access to "Write".

Also give them your BitBucket email address and make sure they share their project with you.

Step 2. Work on your Shop project.

In Code::Blocks or your IDE of choice, create a new project in the same folder as the repository.

Create a **main.cpp** file in the project and open it up.

Begin by writing the basics of your program:

main.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      return 0;
7  }
8
```

We are going to create a class to store food item attributes - price and name - and then implement a menu in main() to let the user select something.

Create the ShopItem class and add its two attributes. (Don't forget to #include <string>!)

main.cpp

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class ShopItem
6  {
7      public:
8      float price;
9      string name;
10 };
```

Now, we'll be able to use the ShopItem class to create items within main().

After that, we'll want to display a menu of foods they can choose from.

Now in main(), we can create and initialize our shop items:

```
12 int main()
13 {
14     ShopItem itemList[3];
15
16     itemList[0].name = "Sandwich";
17     itemList[0].price = 3.99;
18
19     itemList[1].name = "Pizza";
20     itemList[1].price = 9.99;
21
22     itemList[2].name = "Soup";
23     itemList[2].price = 2.99;
24
25     return 0;
26 }
```

This is only initializing.

We will also want to add a program loop, to let the user continue to select items until they decide to quit.

We will also want to display a menu of selectable items each time.

Next, we will add the loop, and output a menu:

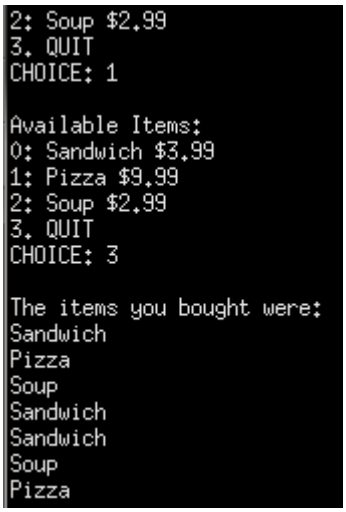
```
24
25      string cart = "";
26      bool done = false;
27      while ( done == false )
28      {
29          cout << endl << "Available Items:" << endl;
30          for ( int i = 0; i < 3; i++ )
31          {
32              cout << i << ": "
33                  << itemList[i].name << " $"
34                  << itemList[i].price << endl;
35          }
36          cout << "3. QUIT" << endl;
37      }
38
39      return 0;
40  }
```

The **while** loop will keep the program running until the user decides to quit, while the **for** loop will display each item in the itemList array for us to select from.

Then, we will let the user choose an option:

```
36          cout << "3. QUIT" << endl;
37
38          int choice;
39          cout << "CHOICE: ";
40          cin >> choice;
41
42          if ( choice == 3 )
43          {
44              done = true;
45          }
46          else
47          {
48              cart += itemList[choice].name + "\n";
49          }
50      } // end of while loop
51
52      cout << "The items you bought were: " << endl;
53      cout << cart << endl;
54
55      return 0;
56  }
```

Now, if we run the program, the user can select options 0, 1, or 2 and the program will continue. If the user selects 3, it will quit, and if anything else is selected, the program will crash. Whee!



This is all we will do with Project B for now. It is time to commit and push our changes for others to be able to work on it.

Open the project folder from the Terminal. We will want to add:

```
hg add main.cpp cpp-class.cbp
```

Here, **main.cpp** is our source file, and **cpp-class.cbp** is the Code::Blocks project file. Change **cpp-class** to whatever you named it.

There are other files in this folder, such as bin/, obj/, and the .exe file or binary if we've built and run the program. We don't want to add these to the repository, as they will just clutter it.

Generally, we wouldn't even add the Code::Blocks project file (*.cbp), but we will for now so the other person can use it. Not everybody who might look at your code will have the same IDE as you, though.

After we add the file, we need to commit. Enter:

```
hg commit
```

Or you could take a shortcut and specify the commit message in the same command:

```
hg commit -m "Commit Message!"
```

Make sure you have a detailed and relevant commit message. Here, “Created ShopItem, basic selection menu” should be OK.

Finally, push your changes to the repository with:

```
hg push
```

Once it pushes (you may need to enter your username/password), you should see the changes on the repository webpage.

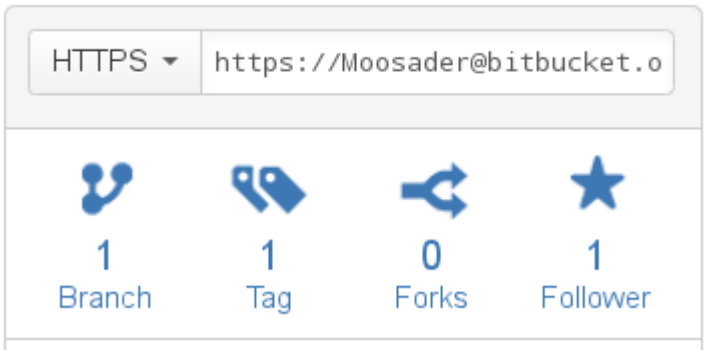
Let the Role A person know that they can start working on the project.

When the Role A person is done with their Project B code, you can go ahead and move on to the next step.

Step 3. Update on Role A's Math project

In BitBucket, if a project is shared with you, it should show up under your Repositories.

Go to the Math project and find the **clone** url, which is marked "HTTPS":



Highlight this text box and copy it.

Create a folder on your harddrive for this project, go to the Terminal, and enter

`hg clone`

with this URL at the end (you can paste by right-clicking).

After the repository is cloned, look for the .cbp file and double-click it to open it in Code::Blocks.

Your task in this repository is to make two new functions - Subtraction and Multiplication.

Copy-paste the existing Addition function and create them with the appropriate math.

Then, in main(), add new examples of using these:

```
28    int sum = Addition( num1, num2 );
29    int difference = Subtraction( num1, num2 );
30    int product = Multiplication( num1, num2 );
31
32    cout << "The sum is " << sum << endl;
33    cout << "The difference is " << difference << endl;
34    cout << "The product is " << product << endl;
35
36    return 0;
```

When you're done, commit your changes and push them:

```
hg commit -m "Added Subtraction and Multiplication functions"

hg push
```

Let your Role A partner know.

Once they let you know that they're done with your Shop project, it's time to grab their changes!

Pulling and Merging Changes

Now that your teammate is done with their changes to your repository, it is time to update!

Navigate back to your **Shop** repository on your harddrive. In the Terminal, type in:

```
hg pull
```

This will pull the changes. Once they're pulled, you'll need to update your repository:

```
hg update
```

There shouldn't be a merge conflict, and you should be good to go.

Open your project again via the .cbp file, build, and run to make sure it still works. Check out the new functionality the other person added!

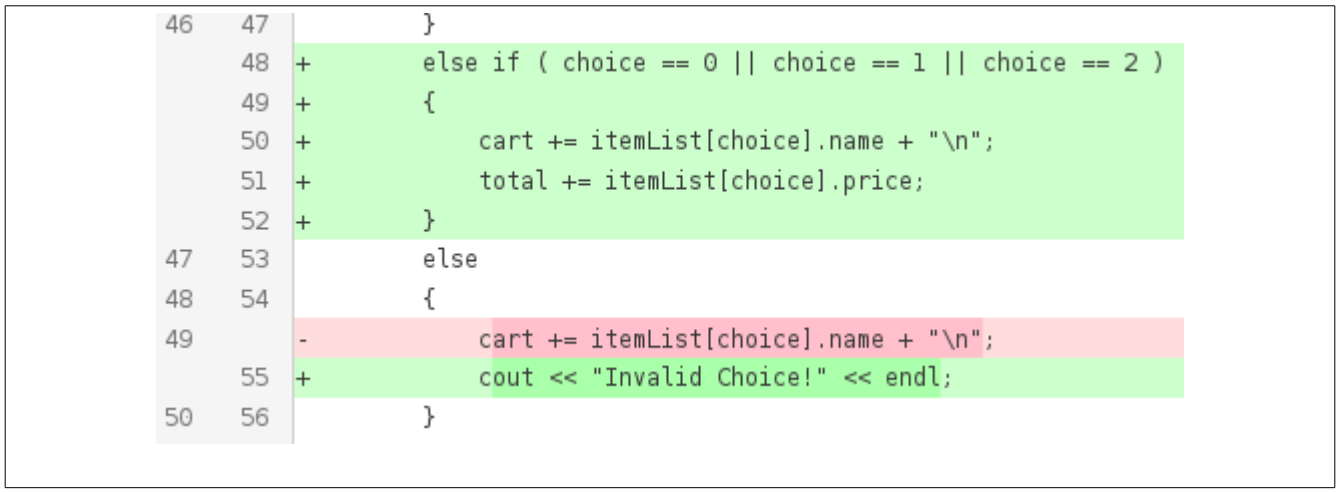
Looking at the Changes in BitBucket

Now that you and your team-mate have both updated your project, go back to BitBucket.

Click on **Commits**, and look for the change where their email address is marked. If you click on the hash code under **Commit**, it will take you to a page with a ***diff***.

A ***diff*** shows you changes made to the project.

The Diff will look something like this:



```
46  47          }
    48  +       else if ( choice == 0 || choice == 1 || choice == 2 )
    49  +       {
    50  +           cart += itemList[choice].name + "\n";
    51  +           total += itemList[choice].price;
    52  +       }
47  53          else
48  54          {
49      -           cart += itemList[choice].name + "\n";
    55  +           cout << "Invalid Choice!" << endl;
50  56          }
```

The lines highlighted in Green were added when the other person committed. If anything is highlighted in Red, that means they removed a line.

Above, it shows the itemList[choice].name line as being removed, but really it was moved into the **else if** statement.

In Closing...

Now you have a BitBucket and a couple of repositories already set up (if you were following along with the Handout!)

If you continue programming in the future, whether in C++ or another language, try to use Source Control to keep track of your changes in a clean, organized manner. Especially if you are currently taking programming classes for school, it will be invaluable!

You can also check out other peoples' repositories for Open Source projects! Doom 3's source, as well as the core Linux code are both on GitHub!

There are also much simpler projects out there to peruse as well!