CS202 Lab 7 - Code Processing

- James S. Plank

- Directory: **~jplank/cs202/Labs/Lab7**
- Last modification date: *Mon Oct 21 17:06:03 EDT 2019*

You and your friends Rad and Banjolina decide to go into business providing web and cell phone support for reward programs like mycokerewards (or, more precisely, like mycokerewards used to be. The current lab is more like Pampers rewards. Mycokerewards got out of the game of redeeming for points, and instead went into sweepstakes. Such is life.) Users can set up accounts with you that will maintain *points*. Users can accumulate points by collecting codes from various products (such as bottlecaps and 12-packs, as in mycokerewards, or such as diapers and wipes in Pampers rewards), and then they can spend the points on various prizes.

Users can enter codes via a web site, or they can register one or more cell phones with their account, and then they can text codes from a given phone number, which will register the points.

Rad is handling the business and marketing end of this endeavor, and Banjolina is doing all of the web programming. Your job is to write the server that maintains information about users, prizes and codes, and talks with Banjolina's web front-end. Since you haven't taken CS360 yet, your server won't do any real networking. Instead, it will communicate via files and standard input.

As with many of our labs, I give you a header file that defines a class, and you have to implement the methods. I have a driver program that you compile with your code, and that will be the final product.

Here's the header, in **include/code_processor.hpp**. Unlike the previous labs, there is no commenting here. I have explanations below.

```
#include <set>
#include <map>
#include <string>

class User {
  public:
     std::string username;
     std::string realname;
     int points;
     std::set <std::string> phone_numbers;
};

class Prize {
  public:
     std::string id;
     std::string description;
     int points;
     int quantity;
};

class Code_Processor {
  public:
    bool New_Prize(const std::string &id, const std::string &description, int points, int quantity);
    bool New_User(const std::string &username, const std::string &realname, int starting_points);
    bool Delete_User(const std::string &username);

    bool Add_Phone(const std::string &username, const std::string &phone);
    bool Remove_Phone(const std::string &username, const std::string &phone);
    std::string Show_Phones(const std::string &username) const;

    int Enter_Code(const std::string &username, const std::string &code);
    int Text_Code(const std::string &phone, const std::string &code);
    bool Mark_Code_Used(const std::string &code);

    int Balance(const std::string &username) const;
    bool Redeem_Prize(const std::string &username, const std::string &prize);

    ~Code_Processor();
    bool Write(const std::string &filename) const;

    void Double_Check_Internals() const;   /* You don't write this */

  protected:
    std::map <std::string, User *> Names;
    std::map <std::string, User *> Phones;
    std::set <std::string> Codes;
```

```
    std::map <std::string, Prize *> Prizes;
};
```

While this looks like a mouthful, it's really not that bad. **Users** store the following data:

- A **username**, which is a one-word name.
- A **realname**, which is the users's full name. For example, my username might be **"jimplank"** and my real name **"Jim Plank"**.
- The total number of points that the user has.
- A set containing the phone numbers registered to the user. Phone numbers are stored simply as strings.

Prizes store the following data:

- An **id**. This is a unique string for each prize.
- A **description**. This is a longer description of each prize.
- The number of **points** that it takes to get one prize.
- The **quantity** of prizes.

A **Code_Processor** keeps track of Users, Codes and Prizes. Users are stored in the map **Names**, which is keyed on their usernames. Phone numbers are stored in the map **Phones**, which is keyed on the phone numbers, and whose **second** field points to the user that has registered the cell phone.

There is a set **Codes**, which stores the codes that have been entered by all users. This set exists so that users can't enter a code more than once. Finally, there is a map **Prizes**, keyed on the id of each prize.

You'll note that both **Names** and **Phones** point to users. In other words, each user has just one **User** instance, and that is pointed to both in **Names** and in **Phones**. If the user has multiple phones, then there will be multiple entries in **Phones** that point to that user. Moreover, there are two data structures that hold phones -- **Phones**, which is keyed on the phone number, and the set **phone_numbers** which is part of the User's data.

Now, you are to write the following methods (I'm omitting the **const** declarations here, to keep the writeup uncluttered. Obviously, you can see the **const** declarations in the header):

- **New_Prize(string id, string description, int points, int quantity)**: This creates a new prize and puts it into **Prizes**. You should return **true** if all is ok. You should return **false** from the following errors without creating anything new:
  - There is already a prize with the given **id** in prizes.
  - **Points** is less than or equal to zero.
  - **Quantity** is less than or equal to zero.
- **New_User(string username, string realname, int starting_points)**: This creates a new user with the given information, and puts it into **Names**. The user will start with no registered phone numbers. You should return **true** if all is ok. You should return **false** from the following errors without creating anything new:
  - There is already a user with that **username**.
  - **Starting_points** is less than zero.
- **Delete_User(string username)**: This should erase the user from **Names**, and it should erase all of the user's phone numbers from **Phones**. After that, it should call **delete** on the user's pointer. You should return **true** if all is ok. You should return **false** if the **username** is not in **Names**.
- **Add_Phone(string username, string phone)**: This should register the given phone string with the user. That means putting the phone on both the **Phones** map, and on the user's **phone_numbers** set. You should return **true** if all is ok. You should return **false** from the following errors without creating anything new:
  - There is no user with that **username**.
  - The phone number is already registered, either with that user or with another.
- **Remove_Phone(string username, string phone)**: This should remove the phone from the system -- both from **Phones** and from the user's **phone_numbers** set. You should return **true** if all is ok. You should return **false** from the following errors without performing any system modifications:
  - There is no user with that **username**.
  - There is no phone string with that **phone**.
  - The phone number is registered to a different user.
- **Show_Phones(string username)**: This should return a string containing all of that user's phone numbers, in lexicographic order, each separated by a newline. There should be a newline at the end of the string too. If the user doesn't exist, return the string

"**BAD USER**". If the user has no phones, simply return an empty string.

- **Enter_Code(string username, string code)**: This is called when a user enters a code. You need to first check the **Codes** set to see if the code has been entered before. If it has, or if the user doesn't exist, return -1. Otherwise, you need to check to see if the code is valid: A valid code's **djbhash()** must either be divisible by 17 or 13. If divisible by 17, then it is worth ten points. Otherwise, if it is divisible by 13, then it is worth three points. If the code is valid, then add it to **Codes**, increment the user's account and return the number of points added. If the code is not valid, simply return zero.

  (BTW, you can use the implementation of **djb_hash** that's in **src/random_codes.cpp**.)

- **Text_Code(string phone, string code)**: This should work just like **Enter_Code()**, except the user's account is identified by the phone number. If the phone number doesn't exist, return -1. Otherwise, this should work just like **Enter_Code()**.

- **Mark_Code_Used(string code)**: This is called to mark a code as used, even though no user is entering it. This is used to help rebuild the server from a saved state (see **Write()** below). If the code is not valid or it is already in **Codes**, return **false**. Otherwise, add it to **Codes** and return **true**.

- **Balance(string username)**: This should return the user's points. If the user doesn't exist, return -1.

- **Redeem_Prize(string username, string prize)**: This is called when a user wants to redeem a prize. The prize is identified by its **id**. If the user or prize don't exist, or if the user doesn't have enough points, return false. Otherwise, decrement the points from the user's account, decrement the prize's quantity by one, and return true. If the prize's quantity is zero, remove the prize from the system (which should involve a **delete** call).

- **~Code_Processor()**: Since **new** is called to create users and prizes, you need to write a destructor that calls **delete** on all the users and prizes. The destructor doesn't have to clear the maps or sets -- that will be done automatically for you when the **Code_Processor** goes away. If you don't understand this point, please ask about it in class.

- I'll describe **Write()** later.

- You don't write **Double_Check_Internals()**. I have written it, and it is in **src/double_checker.cpp**, which you must include when you compile your program. My **makefile** does this for you. The intent of **Double_Check_Internals()** is to make sure that your treatment of phone numbers is consistent, and that you have closed any open files.

---

src/cp_tester.cpp

The program **src/cp_tester.cpp** is a front end for **src/code_processor.cpp**. You call it with filenames on the command line argument, and it reads files that have commands to execute on a **Code_Processor**. If a filename is "-", it reads the commands from standard input. The commands are specified on separate lines -- blank lines are ok, and lines that begin with a pound sign are ignored. Lines may not have more than 20 words on them.

Otherwise, the following commands are supported:

- "***PRIZE*** *id points quantity description*": Calls **New_Prize()** with the given arguments. **Id** is a single word. **Description** may be multiple words.

- "***ADD_USER*** *username starting_points realname*": Calls **New_User()** with the given arguments. **Username** must be one word. **Realname** can contain any number of words.

- "***DELETE_USER*** *username*": Calls **Delete_User** with the given username.

- "***ADD_PHONE*** *username phone-number*": Makes the appropriate **Add_Phone()** call. Both **username** and **phone-number** must be one word.

- "***REMOVE_PHONE*** *username phone-number*": Makes the appropriate **Remove_Phone()** call.

- "***SHOW_PHONES*** *username*": Makes the appropriate **Show_Phones()** call.

- "***ENTER_CODE*** *username code*": Makes the appropriate **Enter_Code()** call. The code should be one word.

- "***TEXT_CODE*** *phone code*": Makes the appropriate **Text_Code()** call.

- "***MARK_USED*** *code ...*": You can specify up to 19 codes on a line. It will call **Mark_Code_Used()** on each of these codes.

- "***BALANCE*** *username*": calls **Balance()** and prints the output.

- "***REDEEM*** *username prize*": calls **Redeem()**.
- "***DOUBLE_CHECK***": calls **Double_Check_Internals()**.
- "***WRITE*** *filename*": calls **Write()** on the given filename. Explanation below.
- "***QUIT***": stops reading. You can simply end input too, and that will stop reading.

Write()

The **Write()** method is very important. Whenever you write a server like this one, you should make it *fault-tolerant*. In other words, you should make it so that it can save its state so that you can terminate the server and start it up again later. The **Write()** method should save the **Code_Processor**'s state to the given file and return **true**. It should return **false** if it can't open/create the file.

The format of **Write()** should be as a file that **cp_tester** can use as input to recreate the state of the **Code_Processor**. It should only consist of **ADD_USER**, **PRIZE**, **ADD_PHONE** and **MARK_USED** lines, and when **cp_tester** is run with the file as input, it should recreate the state of the **Code_Processor**.

I don't care about the order or format of the lines, as long as they create the proper **Code_Processor** when they are fed to **cp_tester**. My grading program will test your files by using them as input to my **cp_tester** and looking at the output of my **Write()** call.

Some examples

Let's start with a very simple example:

```
UNIX> bin/cp_tester -
CP_Tester> ADD_USER tigerwoods 0 Tiger Woods
ADD_USER successful
CP_Tester> ADD_USER the-donald 100 Donald Trump
ADD_USER successful
CP_Tester> PRIZE mp3 40 5000 Free MP3 download from Bapster
PRIZE successful
CP_Tester> PRIZE cancun 10000 1 All expense-paid vacation to Cancun
PRIZE successful
CP_Tester> WRITE cp1.txt
WRITE successful
CP_Tester> QUIT
UNIX> cat cp1.txt
PRIZE     cancun     10000      1 All expense-paid vacation to Cancun
PRIZE     mp3           40   5000 Free MP3 download from Bapster
ADD_USER  the-donald   100 Donald Trump
ADD_USER  tigerwoods     0 Tiger Woods
UNIX>
```

I've added two prizes and two users, and then written the server's state to **cp1.txt**. You'll note that the order of **cp1.txt** is different from my input. That's fine -- if you use it as input to **cp_tester**, it will create the same server state. For example:

```
UNIX> bin/cp_tester cp1.txt -
CP_Tester> BALANCE tigerwoods
0 Points
CP_Tester> BALANCE the-donald
100 Points
CP_Tester> WRITE cp2.txt
WRITE successful
CP_Tester> QUIT
UNIX> cat cp2.txt
PRIZE     cancun     10000      1 All expense-paid vacation to Cancun
PRIZE     mp3           40   5000 Free MP3 download from Bapster
ADD_USER  the-donald   100 Donald Trump
ADD_USER  tigerwoods     0 Tiger Woods
UNIX>
```

When I called **cp_tester**, I gave it two command line arguments: **cp1.txt** and **-**. So, it first read commands from **cp1.txt**, which recreated the same state as when I created **cp1.txt**, and then it read from standard input. When I entered **WRITE cp2.txt**, it created **cp2.txt**, which is identical to **cp1.txt**, since they have the same state.

Suppose I call **cp_tester** with **cp1.txt** and **cp2.txt** on the command line. I should expect four error messages, since the users and prizes already exist when it tries to process **cp2.txt**:

```
UNIX> bin/cp_tester cp1.txt cp2.txt
Prize cancun couldn't be added
Prize mp3 couldn't be added
ADD_USER the-donald unsuccessful
ADD_USER tigerwoods unsuccessful
UNIX>
```

This is because **cp_tester** checks the return values of the **New_Prize()** and **New_User()** calls.

Let's add a few phone numbers and enter some codes. If you check the hashes using **djbhash.cpp** from the hashing lecture notes, you'll see that they are each divisible by 13 and not by 17, so they are each worth three points:

```
UNIX> /home/jplank/cs202/Notes/Hashing/bin/djbhash | awk '{ print $1%17, $1%13 }'
Df18ly81CO1mo4
11 0
IDWNZJ20ENkAxP
2 0
h0yuKnVD6DvRUu
11 0
UNIX> bin/cp_tester cp1.txt -
CP_Tester> ADD_PHONE tigerwoods 865-974-4400
ADD_PHONE successful
CP_Tester> ADD_PHONE tigerwoods 1-800-Big-Putt
ADD_PHONE successful
CP_Tester> SHOW_PHONES tigerwoods
1-800-Big-Putt
865-974-4400
CP_Tester> ENTER_CODE tigerwoods Df18ly81CO1mo4
ENTER_CODE: Added 3 points to tigerwoods.
CP_Tester> TEXT_CODE 865-974-4400 IDWNZJ20ENkAxP
TEXT_CODE: Added 3 points.
CP_Tester> TEXT_CODE 1-800-Big-Putt h0yuKnVD6DvRUu
TEXT_CODE: Added 3 points.
CP_Tester> BALANCE tigerwoods
9 Points
CP_Tester> WRITE cp3.txt
WRITE successful
CP_Tester> QUIT
UNIX>
```

Each **ENTER_CODE** and **TEXT_CODE** call adds three points to **tigerwoods**' account, giving him 9 points in all. After the **WRITE** call, **cp3.txt** looks as follows:

```
PRIZE     cancun     10000      1 All expense-paid vacation to Cancun
PRIZE     mp3           40   5000 Free MP3 download from Bapster
ADD_USER  the-donald   100 Donald Trump
ADD_USER  tigerwoods     9 Tiger Woods
ADD_PHONE tigerwoods 1-800-Big-Putt
ADD_PHONE tigerwoods 865-974-4400
MARK_USED Df18ly81CO1mo4 IDWNZJ20ENkAxP h0yuKnVD6DvRUu
```

The phones have been registered to **tigerwoods**, his point total has been updated, and the codes have been marked as used. Although I put multiple codes on a **MARK_USED** line, you don't have to. Just remember the 20-word limit on a line.

And again, your output does not have to match mine -- it simply needs to create the same **Code_Processor**.

Let's take a look at an example where some prizes are redeemed:

```
UNIX> bin/cp_tester cp3.txt -
CP_Tester> ADD_USER billgates 500000 Bill Gates
ADD_USER successful
CP_Tester> REDEEM tigerwoods mp3
REDEEM:       either the user doesn't exist,
              or the prize doesn't exist,
              or the user can't afford the prize.
CP_Tester> REDEEM the-donald mp3
REDEEM successful
CP_Tester> REDEEM billgates cancun
REDEEM successful
CP_Tester> WRITE cp4.txt
WRITE successful
CP_Tester> QUIT
UNIX> cat cp4.txt
PRIZE     mp3           40   4999 Free MP3 download from Bapster
ADD_USER  billgates  490000 Bill Gates
```

```
ADD_USER  the-donald    60 Donald Trump
ADD_USER  tigerwoods     9 Tiger Woods
ADD_PHONE tigerwoods 1-800-Big-Putt
ADD_PHONE tigerwoods 865-974-4400
MARK_USED Df18ly81CO1mo4 IDWNZJ20ENkAxP h0yuKnVD6DvRUu
UNIX>
```

Since **tigerwoods** only has 9 points, he can't even afford an MP3 from Bapster. **the-donald** has no such problem, and **billgates** can easily afford the Cancun vacation (like he needs it). The updated points for the users and the updated quantities for the prizes have been reflected in the file. Since the quantity of **cancun** went to zero, it has been removed from the system.

---

random_codes.cpp

The program **src/random_codes.cpp** generates random, valid codes.

Strategy

Your strategy here should be to first create a **src/code_processor.cpp** that implements dummy methods for each method. That way you can compile the program and create a **bin/cp_tester**. It won't work (except for **QUIT**), but now you can start programming incrementally.

The first thing you should do is implement **New_Prize()**, and then implement the part of the **Write()** method that creates the file and stores the prizes. Test this by only making **PRIZE** and **WRITE** calls in **cp_tester**.

Then move onto the others. I implemented these in the following order:

- **New_Prize()** and associated **Write()**.
- **Add_User()**, **Balance()** and associated **Write()**.
- **Add_Phone()**, and associated **Write()**.
- **Remove_Phone()** and **Show_Phones()**. With these, I made sure that **Double_Check_Internals()** works, since this is one of those places where it may not work.
- **Redeem_Prize()**.
- **Delete_User()**. Again, I made sure that **Double_Check_Internals()** works here.
- **Enter_Code()** and associated **Write()**.
- **Text_Code()**.
- **Mark_Code_Used()**.
- The destructor.

Although this is a large lab writeup, each of these methods is relatively small. While the grading will of course include the gradescript, the TA's will double-check your destructor by hand.

The Gradescript

The gradescript for this program is a little involved, so let me tell you what it does, so that you can navigate it more easily:

- First, it checks to see if **bin/cp_tester** exists. If not, it's an error.
- Now, let's suppose that we're running gradescript 50.
- It runs the following:

  ```
  UNIX> /home/jplank/cs202/Labs/Lab7/bin/cp_tester /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-*.txt
  ```

- It puts standard output into the file `tmp-050-correct-stdout.txt`.
- It puts standard error into the file `tmp-050-correct-stdout.txt`.
- If any files were created by WRITE commands, they have been named `f0.txt`, `f1.txt`, etc, and they are moved to a new directory called `correct_dir`.

- It runs the following:

  ```
  UNIX> bin/cp_tester /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-*.txt
  ```

- It puts standard output into the file `tmp-050-test-stdout.txt`.
- It puts standard error into the file `tmp-050-test-stdout.txt`.
- Now, it can't compare your files created by WRITE commands to my files, because I'm not requiring your output to match mine exactly. So here's what I do. Suppose we created the file `f1.txt`. I append the line `"WRITE your_dir/f1.txt"` to `f1.txt`, and then I run:

```
UNIX> /home/jplank/cs202/Labs/Lab7/bin/cp_tester f1.txt
```

What that does is create the file `your_dir/f1.txt` with my program, but using your `f1.txt` as input. Now `your_dir/f1.txt` should match `correct_dir/f1.txt` exactly.

- I also test to make sure that your `f1.txt` doesn't have any extraneous lines in it.

So, let's examine gradescript 50. There are two files in the gradescript directory that start with 050:

```
UNIX> ls /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-*.txt
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-B.txt
UNIX
```

Let's look at the first -- as you can see, it adds prizes, users and phones, and it sets a bunch of codes as marked:

```
UNIX> cat /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt
PRIZE     cancun     10000      1 All Expense-Paid trip to Cancun
PRIZE     dogo         570      3 Club Dogo 12-Month Subscription
PRIZE     habitat       35    100 Donation to Habitat for Humanity
PRIZE     silver      1100      2 Two DMC Theatres Silver Experience movie tickets
PRIZE     spinner      750      4 Multi-Function Salad Spinner and Chopper
ADD_USER  ACamelb30  45158 Arianna Camelback
ADD_PHONE ACamelb30  590-448-0257
ADD_PHONE ACamelb30  596-598-2816
ADD_PHONE ACamelb30  702-497-7232
ADD_USER  AChurn40   21934 Audrey Churn
ADD_PHONE AChurn40   235-294-7081
ADD_PHONE AChurn40   361-551-5980
ADD_PHONE AChurn40   597-919-8261
ADD_USER  ADuctil93    882 Anthony Ductile
ADD_PHONE ADuctil93  375-449-4138
ADD_PHONE ADuctil93  509-904-5217
ADD_PHONE ADuctil93  644-036-2649
ADD_USER  AFluenc43    682 Andrew Fluency PhD
ADD_PHONE AFluenc43  495-712-4764
ADD_PHONE AFluenc43  737-246-2569
ADD_USER  AInterv57     13 Aiden Interval
ADD_PHONE AInterv57  081-142-5426
ADD_PHONE AInterv57  183-790-7235
ADD_PHONE AInterv57  855-670-4758
ADD_USER  AJugate14  38987 Austin Jugate
ADD_PHONE AJugate14  174-351-3757
ADD_PHONE AJugate14  610-205-1413
ADD_PHONE AJugate14  856-562-1336
ADD_USER  BBonifa55     93 Brianna Boniface
ADD_PHONE BBonifa55  008-672-3102
ADD_USER  CBarge68   24776 Chase Barge
ADD_USER  DIneffi14  37842 Daniel Inefficient
ADD_PHONE DIneffi14  029-131-8159
ADD_PHONE DIneffi14  462-602-7283
ADD_PHONE DIneffi14  569-485-8923
ADD_USER  GMax14       235 Gabriel Max Set
ADD_PHONE GMax14     556-830-7531
ADD_USER  IParks92     696 Isaac Parks
ADD_PHONE IParks92   119-480-9038
ADD_PHONE IParks92   177-181-8465
ADD_USER  JEcho91    11706 James Echo
ADD_USER  NSvelte62     59 Noah Svelte
ADD_USER  OMauve70    6464 Oliver Mauve
ADD_PHONE OMauve70   120-797-9587
ADD_PHONE OMauve70   364-503-8235
ADD_PHONE OMauve70   451-559-9059
ADD_USER  TVade56        2 Taylor Vade
ADD_PHONE TVade56    355-887-8304
ADD_PHONE TVade56    606-440-6857
ADD_PHONE TVade56    914-780-5061
MARK_USED 6ZzSTTdUaCDy6N 7fClHDlcNlxNpl F3W9dZq4NWCp3F jqivzc4eRM0Jt9 sHDNCtoOnMhctK
UNIX>
```

Now let's look at the second -- it performs a bunch of commands, and writes six files -- `f1.txt` through `f6.txt`

```
UNIX> cat /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-B.txt
REDEEM ACamelb30 cancun
WRITE f1.txt
ADD_PHONE AEmery68 223-558-4601
REDEEM OMauve70 silver
```

```
ADD_PHONE AChurn40 914-780-5061
WRITE f2.txt
WRITE f3.txt
REDEEM DIneffi14 silver
REDEEM OMauve70 dogo
REDEEM AJugate14 spinner
SHOW_PHONES BBonifa55
DELETE_USER NSvelte62
ADD_USER CBarge68 21 James Sanitate
WRITE f4.txt
REDEEM AChurn40 dogo
WRITE f5.txt
TEXT_CODE 914-780-5061 AIV9qdDuoE1Lsz
REMOVE_PHONE ACamelb30 590-448-0257
REDEEM AJugate14 dogo
REMOVE_PHONE ACamelb30 702-497-7232
WRITE f6.txt
UNIX>
```

Let's run my program on it. There is some output, so let's examine it:

```
UNIX> /home/jplank/cs202/Labs/Lab7/bin/cp_tester /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-*.txt
ADD_PHONE: Either the user (AEmery68) doesn't exist,
           or the phone number (223-558-4601) is already assigned
ADD_PHONE: Either the user (AChurn40) doesn't exist,
           or the phone number (914-780-5061) is already assigned
008-672-3102
ADD_USER CBarge68 unsuccessful
UNIX>
```

First, let's verify the first error statement by looking for AEmery68 and 223-558-4601 in the two input files. As you can see, we try to set AEmery68's phone number, and there's no such user:

```
UNIX> egrep 'AEmery68|223-558-4601' /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-*.txt
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-B.txt:ADD_PHONE AEmery68 223-558-4601
UNIX>
```

Let's verify the second error statement. Now you can see that 914-780-5061 was already assigned to TVade56.

```
UNIX> egrep 'AChurn40|914-780-5061' /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-*.txt
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt:ADD_USER  AChurn40   21934 Audrey Churn
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt:ADD_PHONE AChurn40   235-294-7081
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt:ADD_PHONE AChurn40   361-551-5980
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt:ADD_PHONE AChurn40   597-919-8261
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt:ADD_PHONE TVade56    914-780-5061
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-B.txt:ADD_PHONE AChurn40 914-780-5061
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-B.txt:REDEEM AChurn40 dogo
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-B.txt:TEXT_CODE 914-780-5061 AIV9qdDuoE1Lsz
UNIX>
```

The phone number 008-672-3102 was printed out. Let's look for it in the input, and when we find that it belongs to BBonifa55, let's look for BBonifa55. As you can see, we gave the command "SHOW_PHONES BBonifa55", which is what printed "008-672-3102".

```
UNIX> grep 008-672-3102 /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-*.txt
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt:ADD_PHONE BBonifa55  008-672-3102
UNIX> grep BBonifa55 /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-*.txt
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt:ADD_USER  BBonifa55     93 Brianna Boniface
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt:ADD_PHONE BBonifa55  008-672-3102
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-B.txt:SHOW_PHONES BBonifa55
UNIX>
```

Last, there was an error statement: "ADD_USER CBarge68 unsuccessful". Let's grep for CBarge68, and we can see that that username existed and we tried to add it a second time.

```
UNIX> grep CBarge68 /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-*.txt
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-A.txt:ADD_USER  CBarge68   24776 Chase Barge
/home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-B.txt:ADD_USER CBarge68 21 James Sanitate
UNIX>
```

Let's make the directory `correct_dir` and move the six files there:

```
UNIX> mkdir correct_dir
UNIX> mv f?.txt correct_dir
UNIX> ls correct_dir
f1.txt  f2.txt  f3.txt  f4.txt  f5.txt  f6.txt
UNIX>
```

Now, let's run the **cp_tester** in **bin**:

```
UNIX> bin/cp_tester /home/jplank/cs202/Labs/Lab7/Gradescript-Examples/050-*.txt
ADD_PHONE: Either the user (AEmery68) doesn't exist,
           or the phone number (223-558-4601) is already assigned
ADD_PHONE: Either the user (AChurn40) doesn't exist,
           or the phone number (914-780-5061) is already assigned
008-672-3102
ADD_USER CBarge68 unsuccessful
UNIX>
```

The output is identical, so that's good. However `f1.txt` through `f6.txt` don't match the ones in `correct_dir`, because this **cp_tester** implements **Write()** differently:

```
UNIX> ls -l correct_dir/f1.txt
-rw-r--r--. 1 jplank jplank 2020 Oct 22 14:58 correct_dir/f1.txt
UNIX> ls -l f1.txt
-rw-r--r--. 1 jplank jplank 2060 Oct 22 15:07 f1.txt
UNIX> head correct_dir/f1.txt
PRIZE     dogo         570      3 Club Dogo 12-Month Subscription
PRIZE     habitat       35    100 Donation to Habitat for Humanity
PRIZE     silver      1100      2 Two DMC Theatres Silver Experience movie tickets
PRIZE     spinner      750      4 Multi-Function Salad Spinner and Chopper
ADD_USER  ACamelb30  35158 Arianna Camelback
ADD_PHONE ACamelb30  590-448-0257
ADD_PHONE ACamelb30  596-598-2816
ADD_PHONE ACamelb30  702-497-7232
ADD_USER  AChurn40   21934 Audrey Churn
ADD_PHONE AChurn40   235-294-7081
UNIX> head f1.txt
MARK_USED 6ZzSTTdUaCDy6N
MARK_USED 7fClHDlcNlxNpl
MARK_USED F3W9dZq4NWCp3F
MARK_USED jqivzc4eRM0Jt9
MARK_USED sHDNCtoOnMhctK
ADD_USER  ACamelb30  35158 Arianna Camelback
ADD_PHONE ACamelb30  590-448-0257
ADD_PHONE ACamelb30  596-598-2816
ADD_PHONE ACamelb30  702-497-7232
ADD_USER  AChurn40   21934 Audrey Churn
UNIX>
```

So, what I do is append "WRITE your_dir/f1.txt" to the end of `f1.txt`, create the directory `your_dir/f1.txt` and then I run my **cp_tester** on it. If the state of `f1.txt` is correct, regardless of formatting, then `your_dir/f1.txt` should match `correct_dir/f1.txt` exactly (because they were both created by my program):

```
UNIX> mkdir your_dir
UNIX> echo "WRITE your_dir/f1.txt" >> f1.txt
UNIX> /home/jplank/cs202/Labs/Lab7/bin/cp_tester f1.txt
UNIX> ls -l your_dir
total 4
-rw-r--r--. 1 jplank jplank 2020 Oct 22 15:14 f1.txt
UNIX> diff your_dir/f1.txt correct_dir/f1.txt
UNIX>
```

I hope that helps you understand what the gradescript is doing.