**PART 1**

1. Create an environment variable GROUP in the bash shell and set it to either **student** or the name of your alternate group.

Write a C program that does not make use of the **system** function call or shell commands that does the following:

If the value of $GROUP is “student”, create a file exploit in your public\_html directory and change the user permissions of the file so that others can execute the program.

If the value of $GROUP is your alternate group, test whether there already is a file public\_html/exploit that the current user can execute. Make sure that you report the result of the test.

2. Write a function that is passed a player name and an empty struct PLAYER and then fills the userName with the passed player name and randomly generates values for the other fields as they are described in the test. The data structure should be returned automatically to the calling function.

**PART 2**

1. Write and test a framework Command Shell Interpreter in C using readline for input with stub routines that does the following

a. Recognize the following command using a case insensitive compare:

1. hello
2. password *username1 username2 username3 .....*
3. compare *word1 word2 word3 word4 ...*
4. group *file1 file2 file3*
5. display *dbFile recordNo1 recordNo2*
6. replace dbFile *startRecord endRecord*

Note: You must hand in a listing of t

(4 marks)

1. A “test script” for the purposes of recognition is simply a list of the above commands in a file and then use redirection instead of terminal input:

myCmdShell < cmds

Create a 2nd version of your test script with different arguments for the rest of this exercises in #2. This can be done with none of the above working. (2)

1. Respond to the “hello” command by answering hello *username* where the environment variable $USER is used as the response. Change the prompt used by the program after the command is used to the loaded *username* + your original prompt. (2)
2. Implement the password command as follows: (3)
for each username in the list use **getpwnam** to report on the user’s real name, group id and home directory as well as the last access time of their home directory. You may use the stat function as well.
3. Implement the compare command as follows (4)
(The purpose here is it give you some practice with string routines. The fact that readline lets you up arrow and pattern match to previous data entries in the history will help your testing)
4. find the longest word
5. compare word1 and word2 and do a case insensitive comparison and report

which one comes first alphabetically.

1. compare the words but only as far as the 4th character. Report if word1

matches words2 .... And in your test have at least one word that matches

1. Use **strstr** to determine the position of the string representing word1 in words 3

to n. Provide test cases where word1 is absent, present at the beginning and present at the end.

f. Implement the group command as follows: (3)
i. Use **stat** to determine the group id of each file and then use one of the group

functions to determine the group name. Test with your own files using **chgrp** to change the group ownership of one of them and a 3rd file belonging to another user. (Could be root, oracle or, if you recall, every other student in the class should have a .plan and .project file.

1. Implement the display command as follows: (4)
	1. Use the database program you created in Lab 2 to create a database of 100+

records. (If you do not have a database program that worked you may use the data structure struct PERSON and the example program createPeople.c )
(No marks for this)

* 1. The difference between *startRecord* and *endRecord* should be about 5 records. Using a file pointer, open the database file, and then seek to advance to the first record and then read in all the display all records in the list, then close the file. It’s helpful to display the record # to keep track of where you are.

(Note: You can use **gdb** to practice the function calls directly if you are having problems.) (2 marks)

* 1. Using a file descriptor do the same thing but seek to the endRecord and, reading the records one at a time, display the records in reverse order. (2 marks)
1. Implement the replace command as follows: (4)
	1. Display the records from startRecord to endRecord. You should use your

function in g)

* 1. Replace the records from startRecord to endRecord. You should use pwrite

instead of write or fwrite.

* 1. Redisplay the records again by reading them in as a block to a different of

storage than you used to write out the records. (We want to avoid the case of masking a failed read.)