Mustafa A. Hakkoz - 150117509 04.11.2019

Enes Garip - 150116034

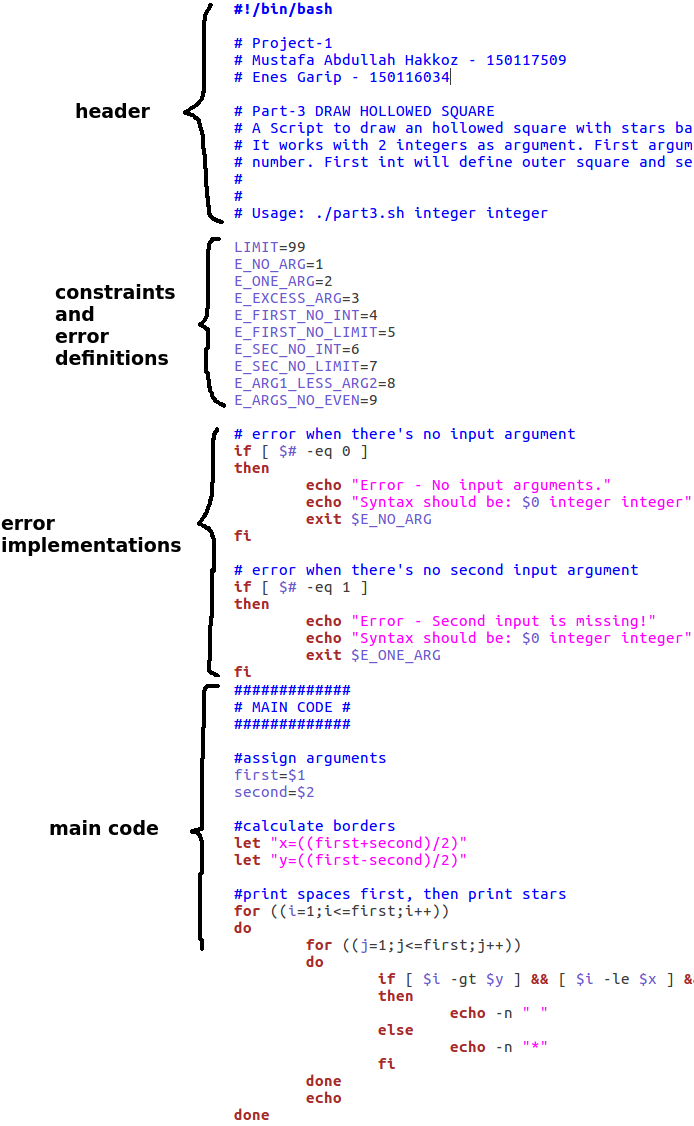
**OPERATING SYSTEMS**

**Programming Assignment # 1 Report**

### Subject

This project consists of 5 different parts plus 1 extra part of bash scripting about string operations, globbing, file operations, basic text editing etc.

* **Folder Content**

There are 5 “.sh” files for each part of project in the zip folder:

“part1.sh”, “part2.sh”, “part3.sh”, “part4.sh” and “part5.sh”

And 1 more “.sh” file for main menu:

“menu.sh”

You can run each file by themselves by typing their names and corresponding argument in the terminal. For example:

´**$** *path/filename.sh arg1 arg2´*

Or you can run “main.sh” without arguments and it will redirect you other 5 parts.

´**$** path/menu.sh´

* **Implementaion Details**

While writing scripts we sticked to some format.

- Each script have its on header which consists student information, project name, description and usage.

- Except for menu.sh all scripts have error definitons and error impelementations that that examine given cases in project document.

-And lastly, all scripts have main code part for main algorithms.

**1. Check for palindromes “part1.sh”:**

This is a bash script to check if a given string is a polindrome or not. It works with single words w/wo quotes as input. Remark that, when it comes to sentences with many words it requires to use it with quotes to prevent multi arguments. For example:

´**$** *./part1.sh "string with many words´*

For main code, we first convert the input string to all lowercase, then clear the spaces. After calculating the middle position, we take reverse of string and compare it to original one character by character until midpoint. If they are equal, then string is polindrome.

**POSITIVE SUCCESFUL CASES:**

* **Single word:** Try it for “madam”
* **Sentence:** Try it for “ey edip adanada pide ye”



**NEGATIVE SUCCESFUL CASES:**

* **Single word:** Try it for “deneme”

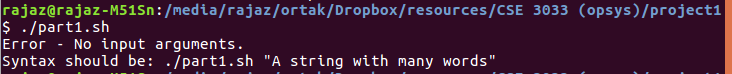


* **Sentence:** Try it for “Bu bir deneme cümlesidir”

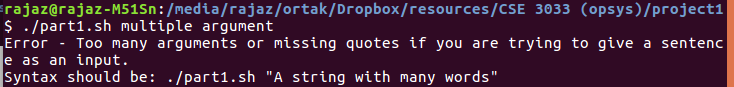


**ERROR CASES:**

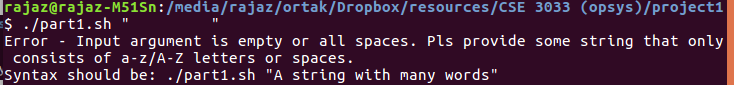
* **When there's no input argument:** Try it without any argument.

****

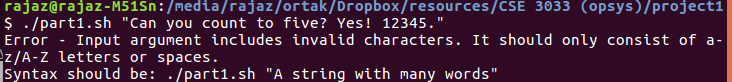
* **When there's more than 1 input argument:** Try it for the string “multiple argument”

****

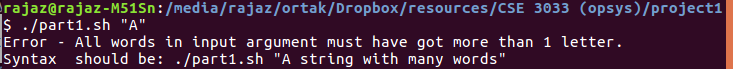
* **When input argument is empty string:** Try it for " "

****

* **When input argument is NOT string with a-z letters:** Try it for

"Can you count to five? Yes! 12345."

* **When input argument has exactly 1 letter:** Try it for “A”



**2.Move .c files “part2.sh” :**

This is a bash script to find all the files whose extension is ".c" in a given directory and move them into newly created "cprogs" directory. It works with an optianal pathname as argument. If user doesn't provide a pathname, program will run under current working directory. Usage:

´**$** .*/part2.sh path(optional)´*

For main code, first we assign input argument to a variable “directory”. if it isn't provided, assign current directory to the variable. After using “cd $directory” we create “cprogs” folder under it and finally move all files with “.c” extension to it. If there’s already a folder named “cprogs” under given directory, mkdir command will take care of the situation automatically so we didn’t put a special error case for it.

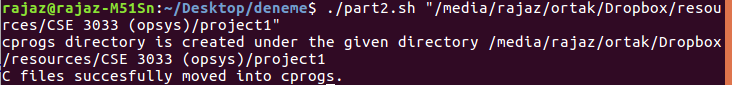
**SUCCESFUL CASES:**

* **Current directory:** Try it for no argument.



* **Optional path:** Try it with input argument“/media/rajaz/ortak/Dropbox/resources/CSE 3033 (opsys)/project1”

Remark that argument can work with spaces and special characters if it is placed between quotes. So we should’t forget to put quotes for all cases.

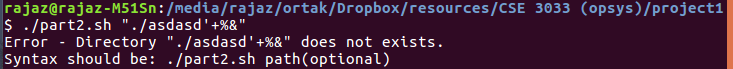


**ERROR CASES:**

* **When there's more than 1 input argument:** Try with 2 arguments "./" and "/media"



* **When given path doesn't exist:** Try with an invalid path like "./asdasd'+%&"



* **When there's no .c file in the directory:** Try with a directory which consists any .c file.



**3. Draw hollowed square “part3.sh” :**

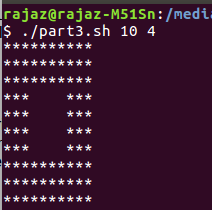
This is a bash script to draw an hollowed square with stars based on 2 numbers given. It works with 2 integers as argument. First argument must be greater than the second one, also their difference must be even. So that first int will define outer square and second int will define inner square. Usage:

´**$** *./part3.sh integer integer´*

For main code, we first assign input arguments to variables and by using them we calculate “x” and “y” which are related to border thickness. Through them, in the inner loop, we first draw spaces then stars. This is the way easier then writing conditionals for inverse case.

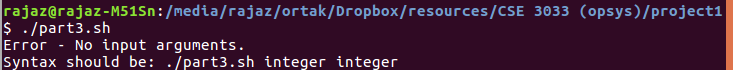
**SUCCESFUL CASES:**

* **Thick border:** Try for 10 and 4 ✓ **Thin border:** Try with 9 7

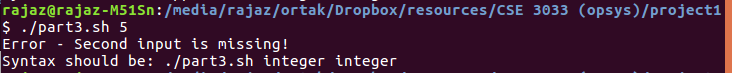
**

**ERROR CASES:**

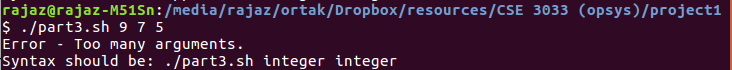
* **When there's no input argument:** Try it without any argument.

****

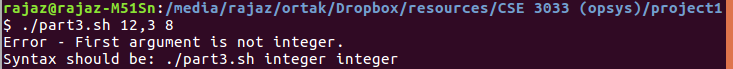
* **When there's no second input argument:** Try it with only 1 argument.



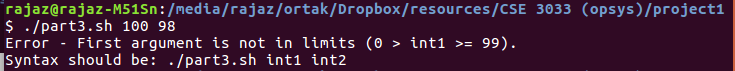
* **When there's more than 2 input argument:** Try with “9 7 5”



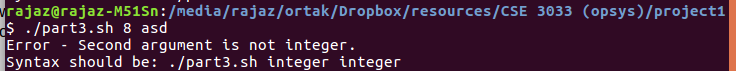
* **When first argument is not integer:** Try with “12,3 8”



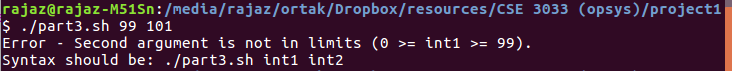
* **When first argument is not in limits (<1 or >$LIMIT):**  First argument should be greater than 1 and upper limit is defined as 99 in our script. So try with “100 98”



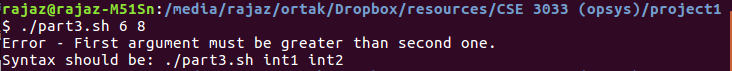
* **When second argument is not integer:** Try with “8 asd”. Remark that program only accepts positive integers, neither floats nor alpha / punct characters.



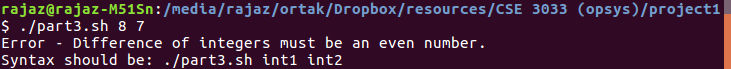
* **When second argument is not in limits (<0 or >$LIMIT):** Second argument should be greater than 0 and upper limit is defined as 99 in our script. So try with “99 101”



* **When first argument not greater than second one:** Try it for “6 8”



* **when their difference not an even number:** Try it for “8 7”



**4. Uppercase conversion “part4.sh” :**

This is a bash script to convert matching characters to uppercase in all text files in a given directory. It works with a wild card and an optianal pathname as arguments. Usage:

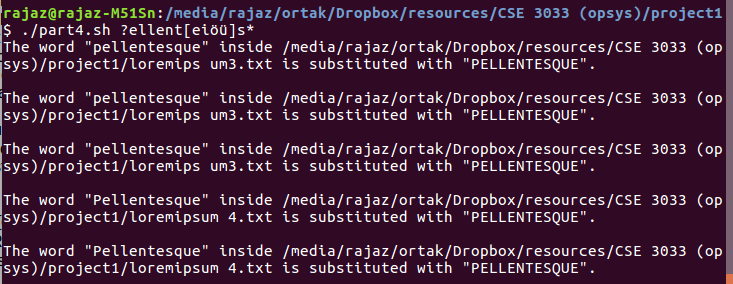
*´***$** *./part4.sh "wildcard" path(optional)´*

For main code, we first clear spaces from first argument and assign it to the variable “wildcard”. Also we assign scond argument to the variable “directory” if it s provided; otherwise we assign $PWD to “directory”.   
  
Then for every text file in the “$directory” we read them line by line. We construct an another inner loop to read every word in “$line” and clear punctiations. After all that we compare words to “$wildcard”. If they match, we print the original word in uppercase form. If they don’t match we print the original word without changing. After reading loops we write whole printing to a file named ”.file\_content” which will be written on to the original file with .mv command so that the original file will be replaced with the file “.file\_content”.

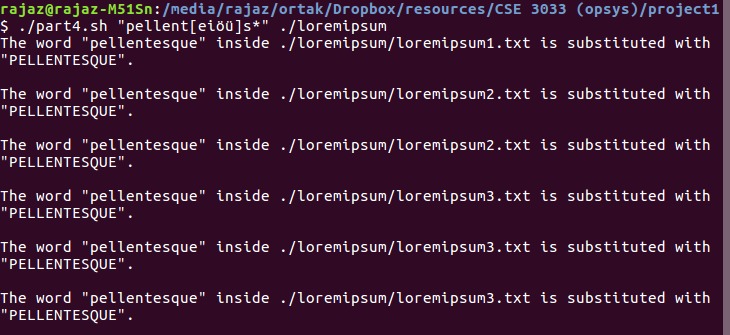
For an extra detail, to exclude screen output to file output, we write screen outputs in to different file named “.output”. And finally, after all implementation we show the content of “.output” on the screen then remove it, if it exist in first place.

**POSITIVE SUCCESFUL CASES:**

* **Only 1 argument:** Try it with wildcard “?ellent[eiöü]s\*” to show the code can work with all symbols in standard wildcard syntax. Remark that program can work with filenames and pathnames which consist whitespace. But again don’t forget to put “pathnames with spaces” between qutoes if you are giving them to program as second argument.

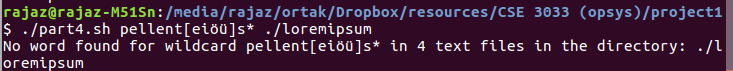


* **With a given path:** Try it with “pellent[eiöü]s\*” and “./loremipsum”. Remark that program is CASE SENSITIVE to not break the functionality of wildcards so this output only cathces lowercase “p” unlike the above.



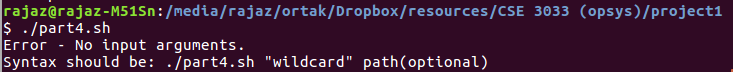
**NEGATIVE SUCCESFUL CASES:**

* **When there’s no matched word found:** If we just run the previous code again, it doesn’t find any word since they are all converted into uppercase already. Try it again with "pellent[eiöü]s\* ./loremipsum”

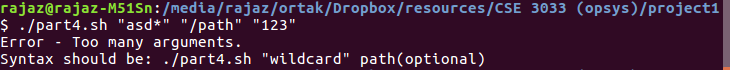


**ERROR CASES:**

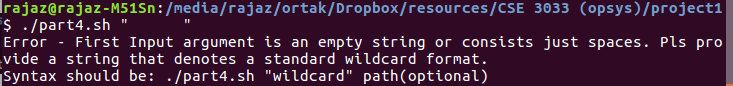
* **When there's no input argument:** Try it without any argument.



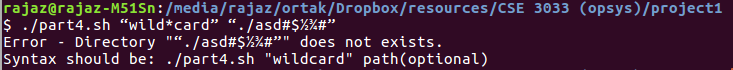
* **When there's more than 2 input argument:** Try it with “asd\*” “/path” “123”



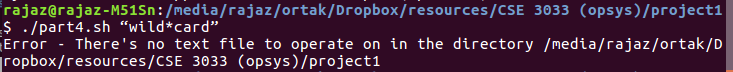
* **When first input argument is empty string:** Try it with “ ”



* **When path doesn't exist:** Try it with “wild\*card” “./asd#$½¾#”



* **When there's no .txt file in the directory:** Try it witf any folder that doesn’t contain text files. For example “wild\*card”



**5. Delete files “part5.sh” :**

This is a bash script to find all the files whose size is zero under a given directory and ask the user to delete them one by one. It works with an optianal pathname as argument and -R option which yields recursive search for sub directories. If user doesn't provide a pathname, program will run under current working directory. Usage:

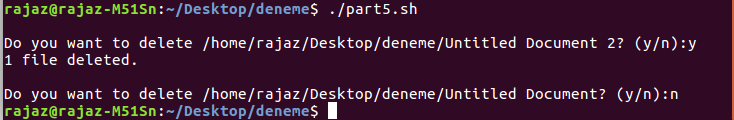
´**$** *./part5.sh -R(option) path(optional argument)´*

For implementation, we first assign input arguments to corresponding variables “$option” and “$path” for 3 different situtations (only “-R” provided, only path provided, both of them provided). Then we build an array named “$files” by using “find ./ -type f -size 0” command. It does recursive search in directory and only looks for 0 sized files. If “-R” option doesn’t provided by user, then we add an extra criteria “-maxdepth 1” to the command for searching only on current directory.

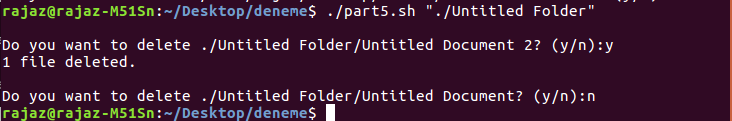
After building $files array, for every file in it we ask to user if he wants to delete the file or not, then read his answer, and finally remove the “$file”, if he types “y”.

**POSITIVE SUCCESFUL CASES:**

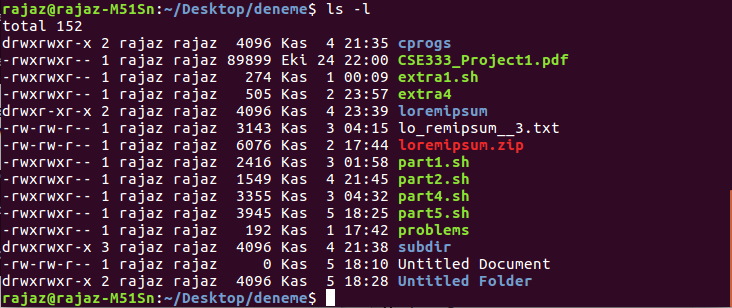
* **With no arguments:** Try it without any argument.



* **With only path argument:** Try it with any folder that may contain zero sized file. For example “./Untitled Folder”. Remark that, when path contains a whitespace you should put pathname between quotes.

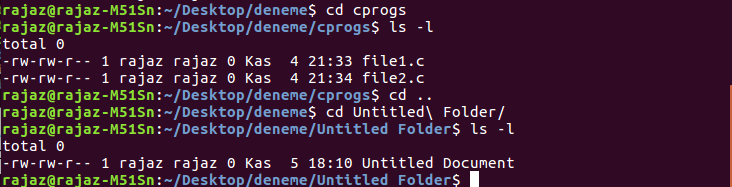


* **With only “-R” option:** First type “ls -l” to see files and subfolders.

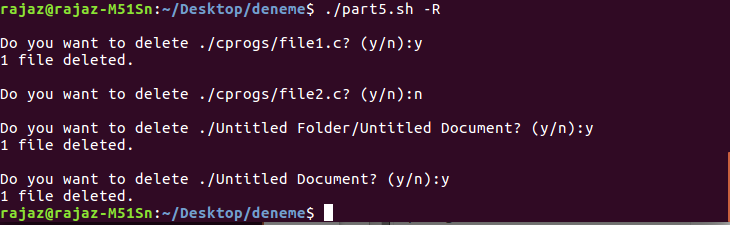


There’s only 1 zeo sized file: “Untitled Document”

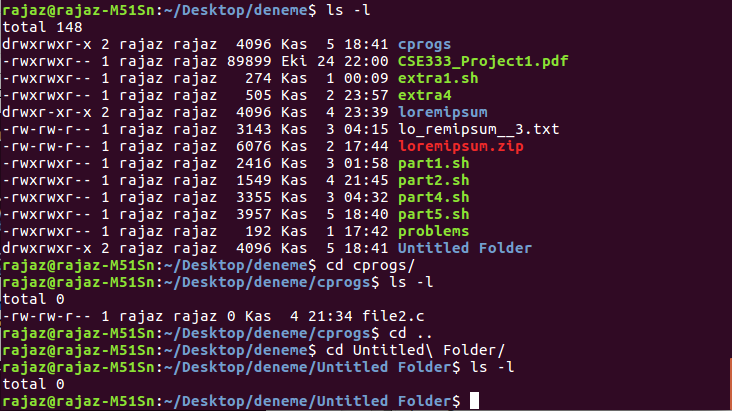
And under subfolders “cprogs” and “Untitled Folder” there are more zero sized files.



There are 3 zero sized files in subdirectories and 1 zero sized file in parent directory. Now run the program with only “-R” option. (Don’t forget to go to program’s directory before run it.)

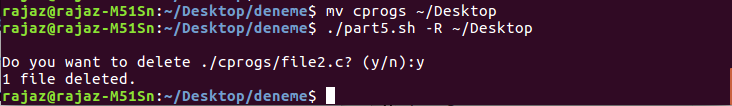
****

Program asks for all 4 of them. We accepted to remove them except “file2.c” under “cprogs”. Let’s check current situtation for main directory and other 2 subdirectories:

****

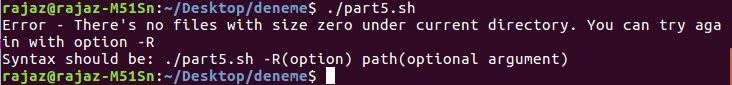
As you can see above, 3 files we accepted are removed succesfuly. Only “file2.c” under “cprogs” is left.

* **When 2 arguments provided:** Let’s try to remove “file2.c” under “cprogs”. Move “cprogs” to Desktop then run the code with “-R” and “~/Desktop”. (tilda is parent directory of Desktop).



**NEGATIVE SUCCESFUL CASES:**

* **When there’s no zero sized file under current directory:** Try it without arguments.

****

* **When there’s no zero sized file under all sub-directories:** Try it with “-R”

****

**ERROR CASES:**

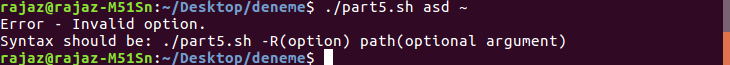
* **When there's more than 2 input argument:** Try it with arguments “-R” “~” “asd”
* **When 2 input arguments are given but reverse order:** Try it with “~” and “-R”

****

* **When 2 input arguments are given, first is valid path but second is NOT "-R":** Try it with “~” and “asd”

****

* **When 2 input arguments are given and first is NOT “-R:** Try it with “asd” and “~”

****

* **When 2 input arguments are given, first is “-R” but second is invalid path:** Try it with “-R” “asd”

****

* **When only 1 input argument is given and it’s invalid path:** Try it with “asd”

****

* **When only 1 input argument is given and it’s NOT a valid option (only detects a letter after hyphen) :** Try it with “-r”. Remark that option is case sensitive so -r will not work.

****

**6. Menu “menu.sh” :**

This is a bash script to create simple menu and take action according to that selected. It doesn’t take any argument. To use it only type:

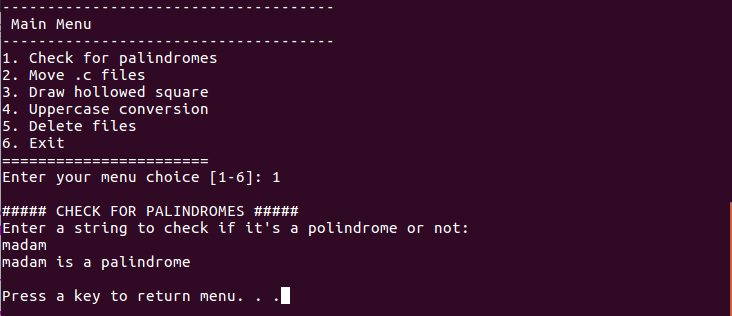
`**$** ./*menu.sh`*

For implementation, we first print menu items to screen in a while loop and ask for choices. When user types a number between 1-5, program redirects user to corresponding script. But just before run the script it asks for input arguments. If arguments are optional, program wants user to hit enter and to submit an empty string.

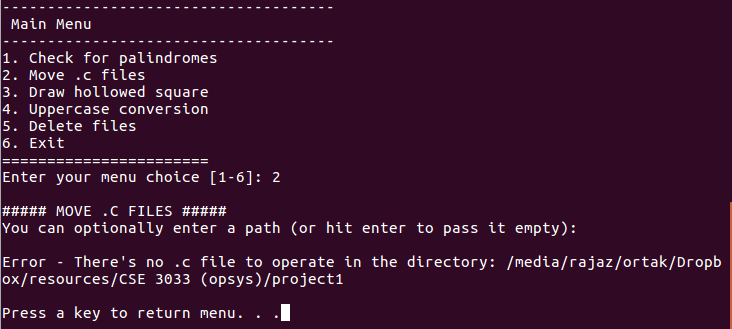
After task completed, menu program will keep running so it will keep asking for new choices unless user enters “6” and terminates the program.

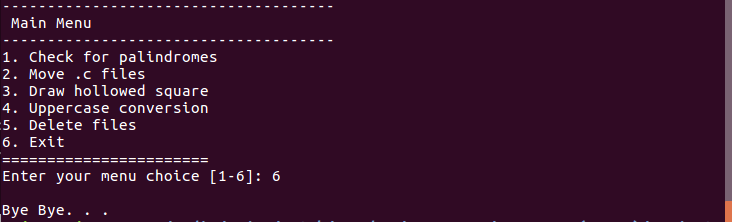
**POSITIVE SUCCESFUL CASES:**

* **When user choose and complete a task succesfuly:** Run the menu, then type a number between 1-5 and hit enter. Then you can submit arguments if necessary.

****

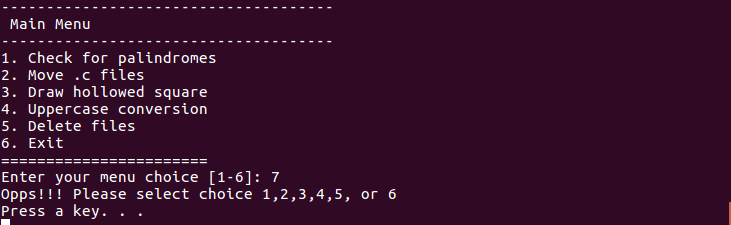
* **When user choose a tast but it returns an error:** Type a number between 1-5 and hit enter. Submit arguments (in the case above, argument was optional so we just hit enter without writing anything), if tast returns an error program will print it.

****

* **When user choose exit:** Type “6” to see it.

**ERROR CASES:**

* **When user types characters rather numbers between 1-6:** Type “7” to see it.

****