INTRODUCTION

This project focuses on various tools developed for managing exams. These tools hold functionalities such as creating exams, reading exam data, editing exam questions, appending additional questions to exams, deleting specific questions from exams, and conducting exams. To enhance the appearance and usability of the console interface, a dedicated header file named "projUI.h" has been coded. Let us go through each of the files to get a better explanation.

createExam is a tool for creating questions and different choices and compiles it into an exam. Upon execution, the program first checks the command-line arguments. If the user provides incorrect arguments, it displays an error message in a red dialog box and prompts the correct syntax. If a file with the provided name already exists, the program prompts the user whether they want to overwrite it. If the user declines, the program terminates; otherwise, it proceeds with creating the exam file. The user is then presented with a graphical interface using colored dialog boxes. They can input exam questions, choices, and answers interactively. Each question is represented by a structure examStruct, containing the question, choices, and correct answer. The program dynamically allocates memory for an array of examStruct to store multiple questions. The user can terminate the question input process by typing "/stop". Once all questions are inputted, the program writes the total number of questions (N) followed by the array of examStruct into the exam file using binary mode. It then deallocates the memory and displays a confirmation message in a green dialog box.

readExam is a tool for reading questions created by createExam. When executed, it first checks the command-line arguments to ensure that the user has provided the correct syntax. If the file specified by the user doesn't exist, it prompts an error message indicating the absence of the file. If the file exists, the program reads the total number of questions stored at the beginning of the file and dynamically allocates memory to store the questions and their corresponding choices.

It then reads the file's contents into the memory that has been allotted. It then goes through each question, showing the answers on the screen along with the choices they made. The file is closed and the allotted memory is released once all of the questions have been read.

appendExam’s purpose is to append questions to the next questions, it behaves just like createExam, yet the main purpose is to add other questions. Assuming the file is there, it uses a examStruct to dynamically allocate memory to contain the number of exam questions that are currently available. appendExam asks the user to enter a new exam question inside of a loop, then displays its four options (labeled A, B, C, and D). It is expected that the user will input '/stop' to signal the end of input. The program dynamically reallocates memory to fit the new inputs for each one. The application adds the updated number of questions to the file after receiving all new questions, then appends the new questions and answers. Lastly, it closes the file, frees up the memory that was allotted, and shows the user a confirmation message.

editExam is used for editing a certain question within the exam file. Upon execution, it checks the command-line arguments to ensure the correct syntax is followed, displaying an error message if not. It then reads the specified file to retrieve the number of questions and their details into memory. Then the user is asked to re-enter the question and the choices. After that, the question will be modified and saved back to the file. When it is completed, it will provide a confirmation in a dialog box which is a function to “projUI.h”.

deleteqExam is designed to delete a specific question within the exam file. When it is executed, it reads the file in “rb” (binary) and checks for any error in syntax, allocation and whether it exists. The program reads the number of questions and stores it back to variable N. Then it allocates the memory dynamically to adjust for deletion, and saves the modified data back to the file.

doExam is the tool to conduct exams created by createExam. It is also compatible with the usage of editExam, deleteqExam, and appendExam to change the exam file. When it is called, it checks first if the syntax is correct. Then it asks the user for the important details, such as Student Number, Course Program, First Name, and Last Name. After that, it reads the exam data and randomizes the questions. It now prompts the question and choices to the user. The user will then need to answer the question. After answering, it will prompt if the user’s answer is correct or incorrect, and increments or retains the score depending on the correct answer. After answering, it divides the N (the number of questions) by 2 and compares the user’s answer, and prompts whether they passed or failed. It will then store the scores on the examScores.txt, and close the file pointers and frees the array.

And last, the projUI.h. This header file is created externally to compensate and improve the design of the console by modifying the colors and creating dialog boxes. The functions included in this header file are createDlgBox, setPalette, checkLength, and clearScreen. The purpose of createDlgBox function is to create dynamic dialog boxes within the console. When I (Raven) thought about creating a dialog box to be used on errors, confirmations and prompts, the first problem was making the dialog box dynamic to accommodate the prompt and the subject (e.g., filename). So, I thought creating a function to verify the character length of the prompt and the subject, and that was the purpose of checkLength. After checking the length which is stored on the variable totalCharLength, it will be the basis for the rows which in the first and last column, will print a dash character “-“, and in the 2nd and 4th column will print a vertical line “|” in the beginning and the end row, and prints whitespace within the rest, and in the middle line, the prompt and the subject will be printed out.

The syntax of createDlgBox function is:

createDlgBox(char \*subject, char \*prompt, char \*punctuation, char \*arrangement)

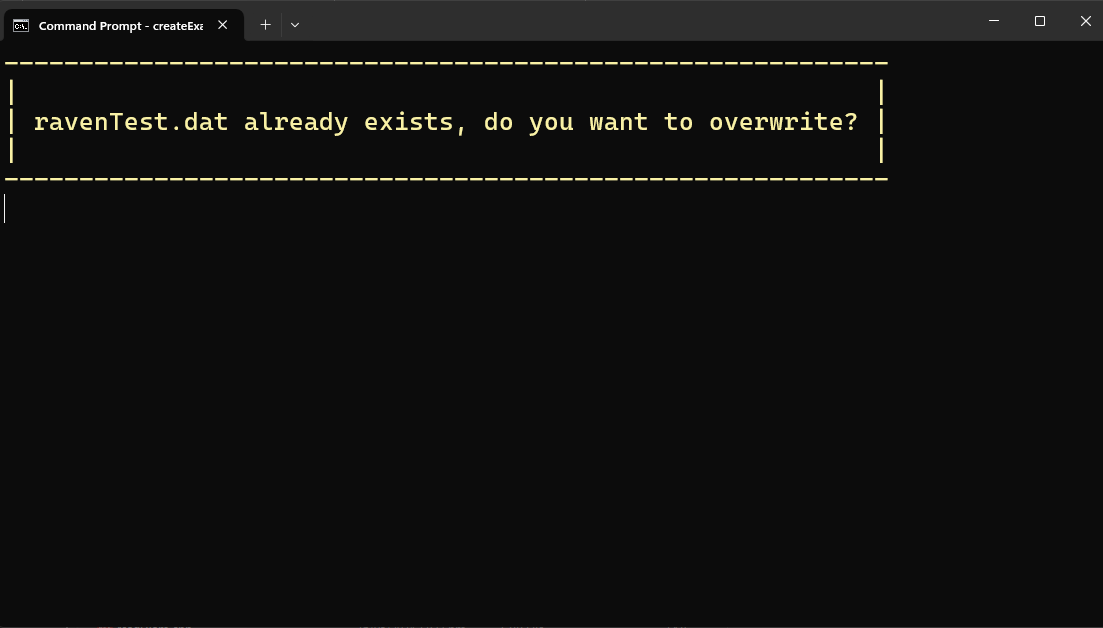
The subject could be an argument vector passed on the command line, such as a filename, question to be edited/appended, etc. The prompt parameter is anything you would want to prompt to the user, may it be an error or a confirmation. The punctuation parameter is needed to assure when a prompt is asking or declaring. And the arrangement parameter is crucial because there are cases when a subject should be printed first before the prompt, and vice versa. The proper arrangement parameters are “sp” (Subject then Prompt) and “ps” (Prompt then Subject).

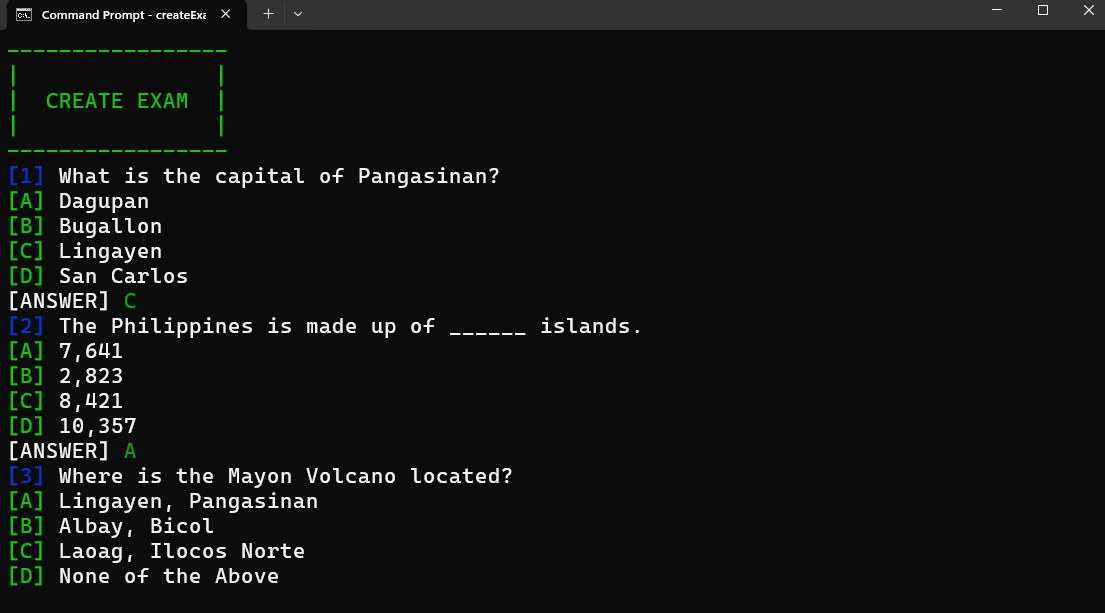
“sp” = “testExam.dat already exists, do you want to overwrite?”

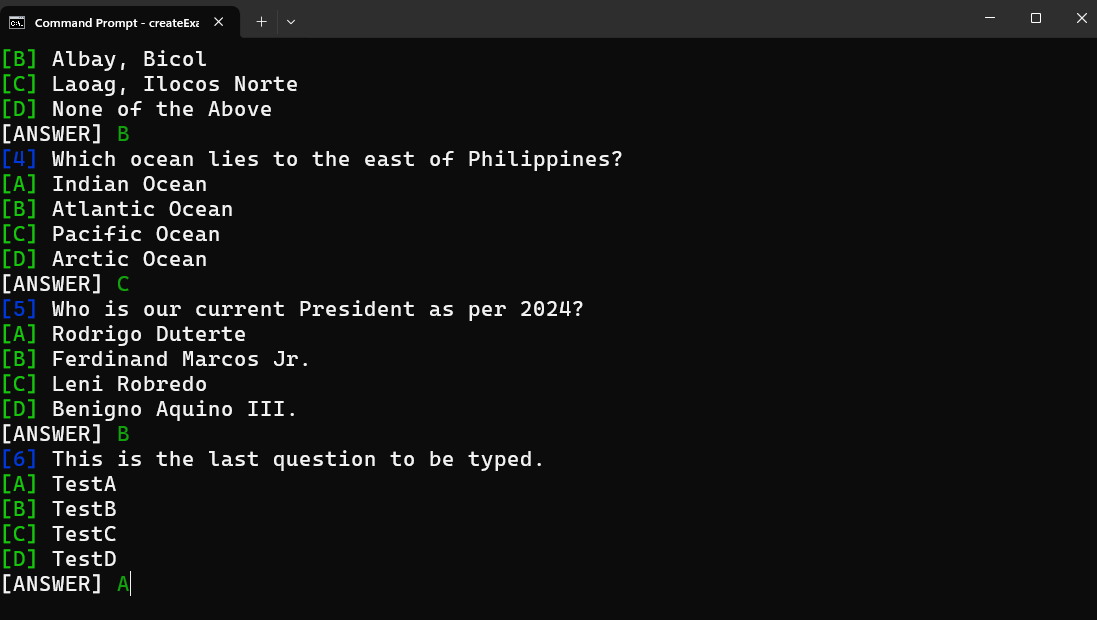
“ps” = “Changes are saved to testExam.dat”

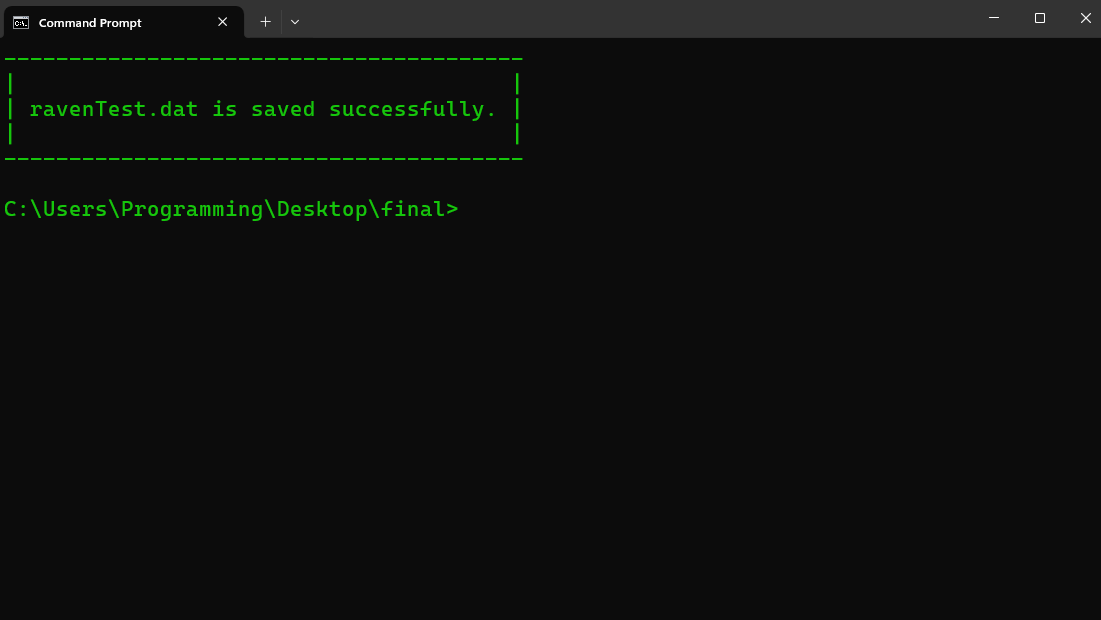
The next function aside from mentioned is the setPalette. Instead of using SetConsoleTextAttribute(color, <color\_values>) and inputting the color values within the function every time I want to change the text color, I created the setPalette(<color>) that accepts string datatype and would be easier to use. It converts the string to an integer color value and changes the console text color immediately. Reference of color values is from cplusplus.com/forum/beginner/54360/. The clearScreen function is just an invoking a DOS command called “cls” which clears the console. This is important to make the design look cleaner.

createExam Code Output Samples

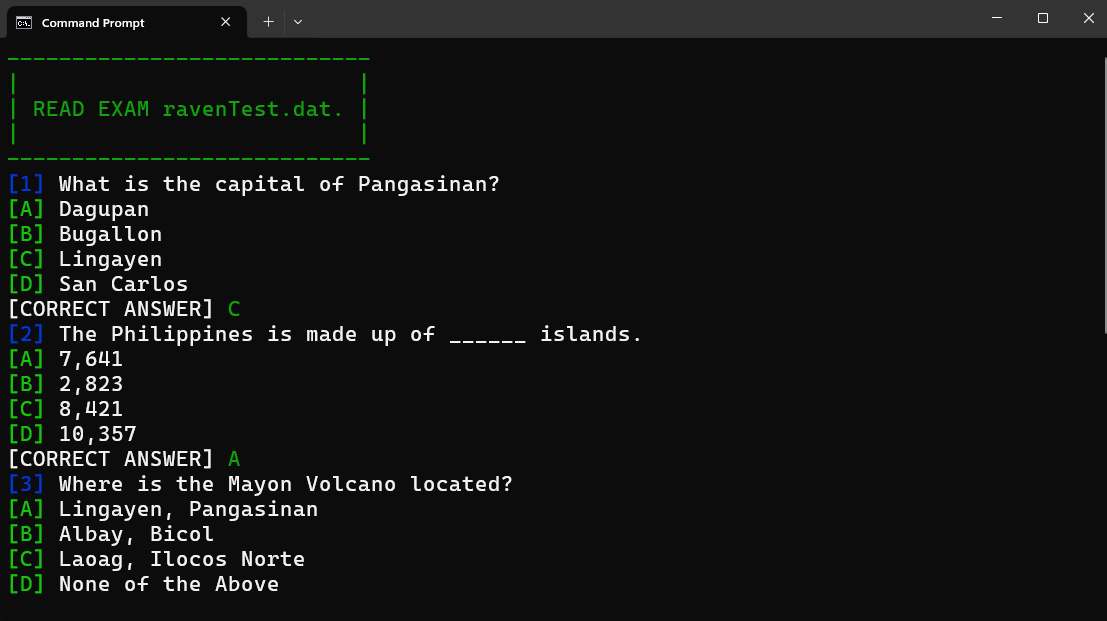


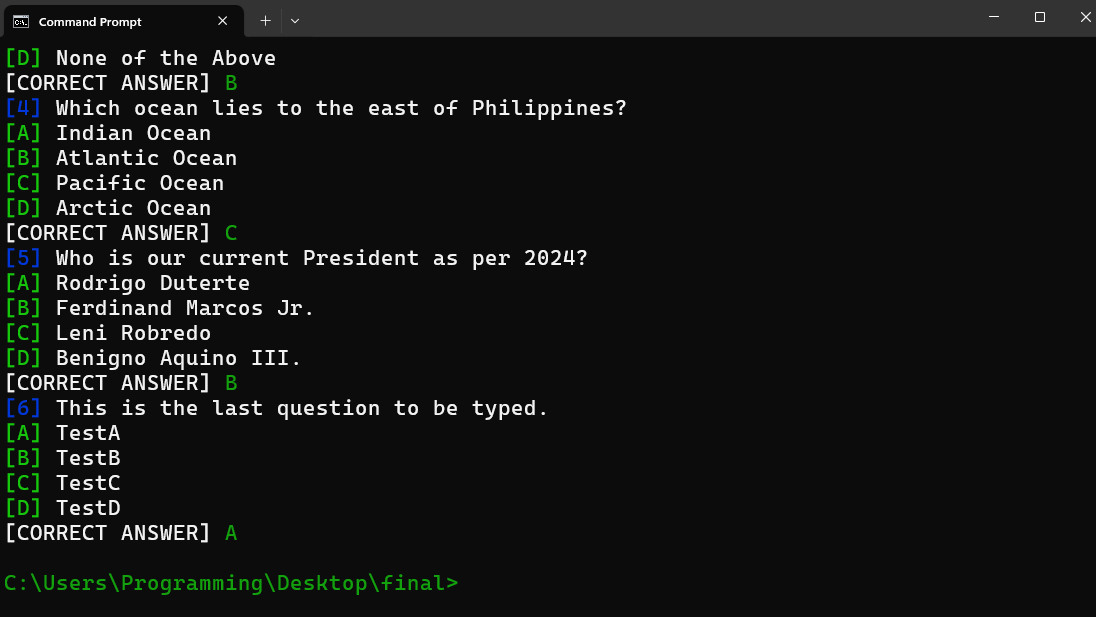


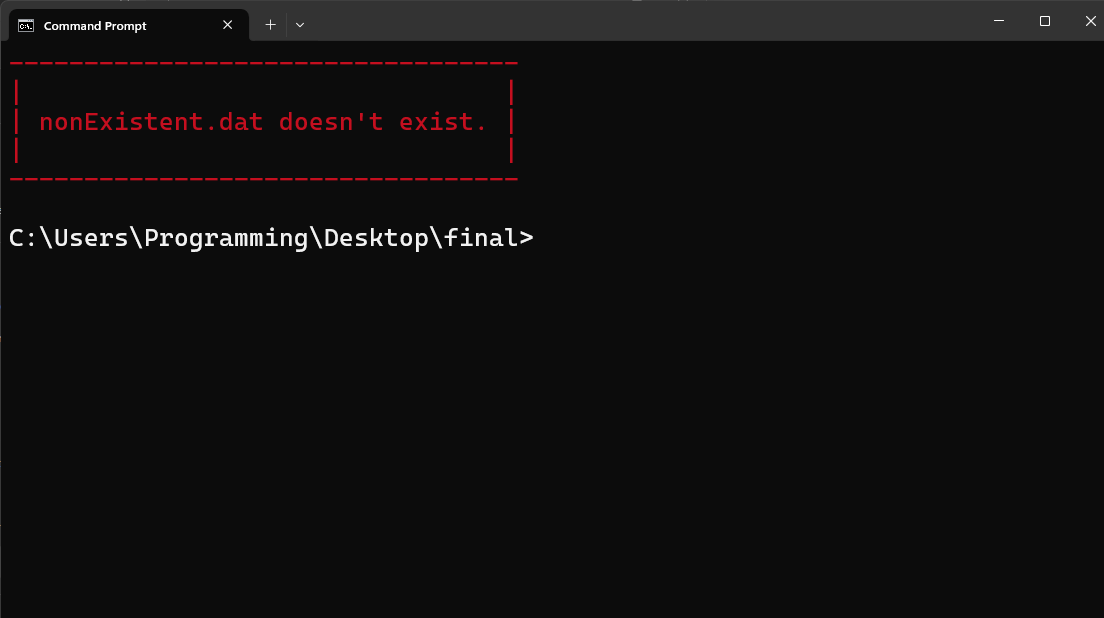




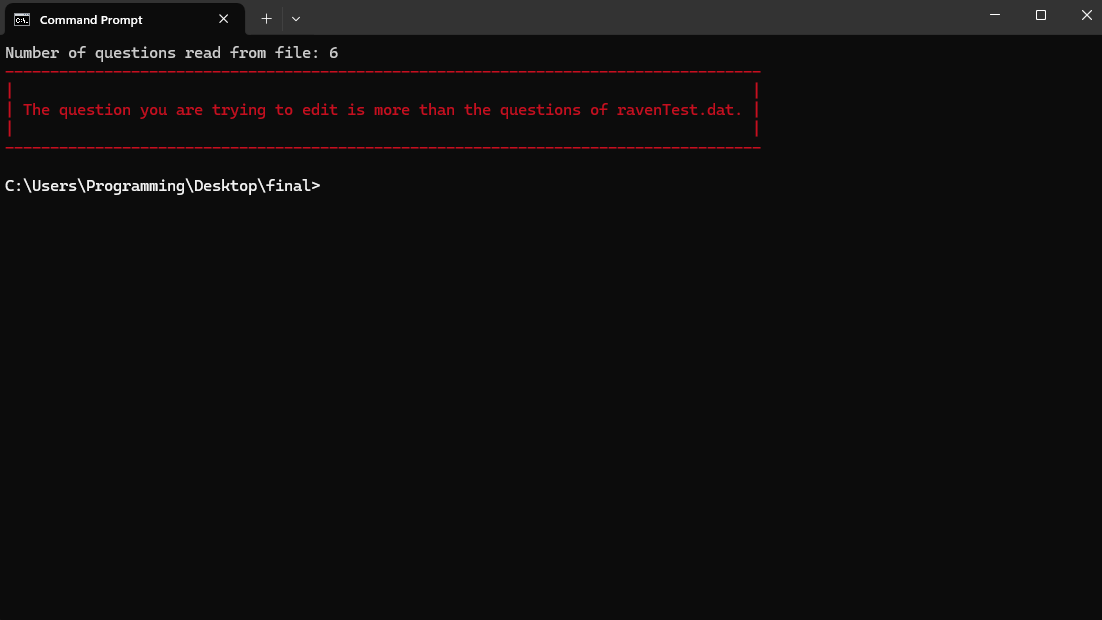
readExam Code Output Samples

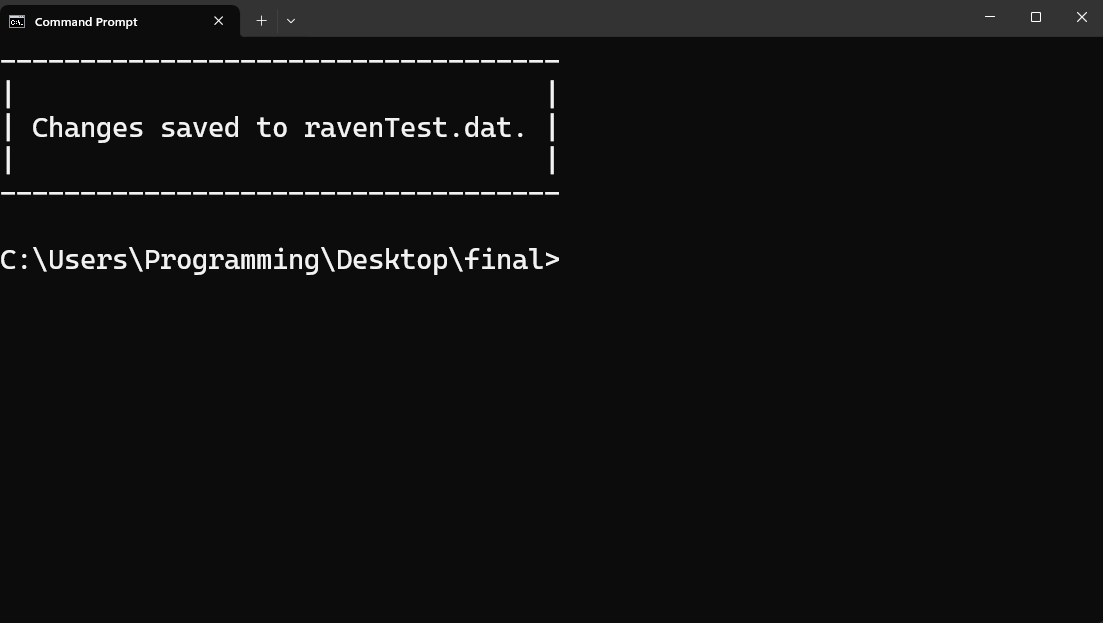
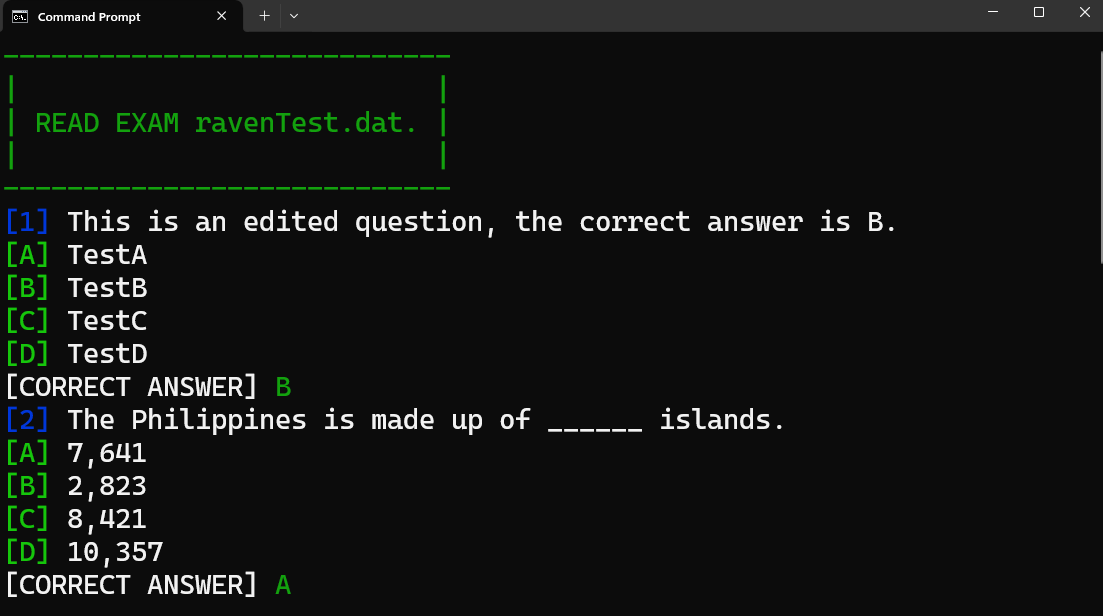
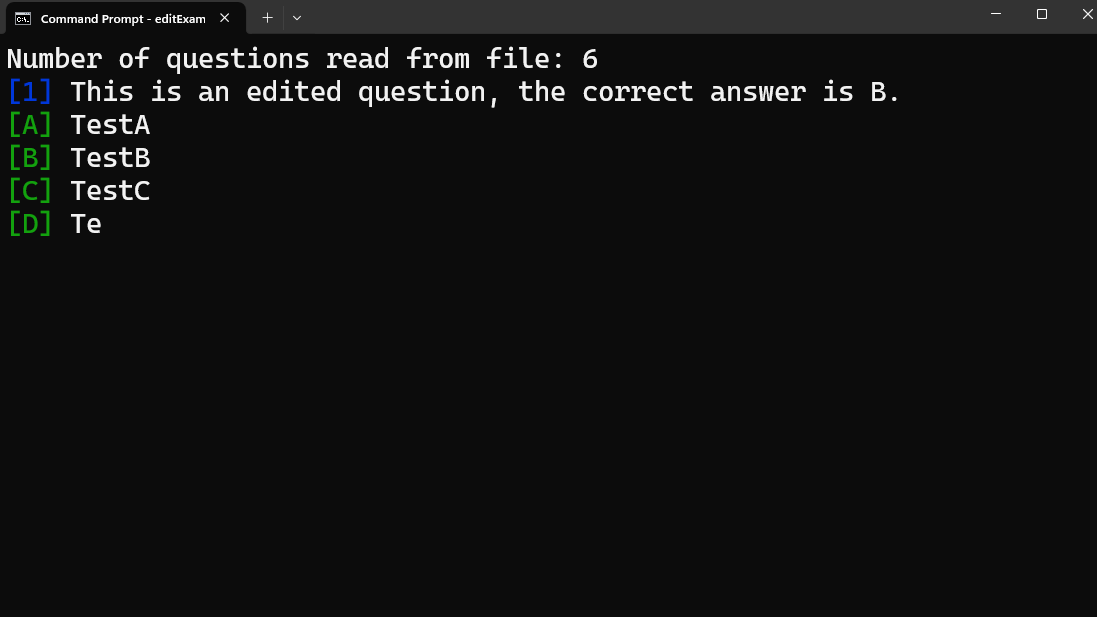




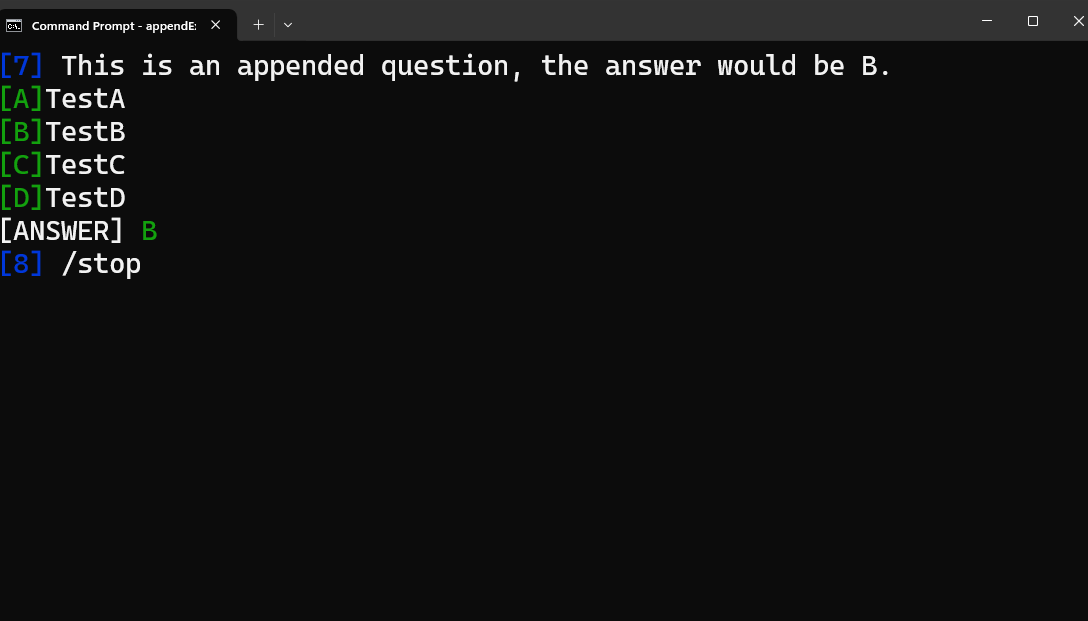
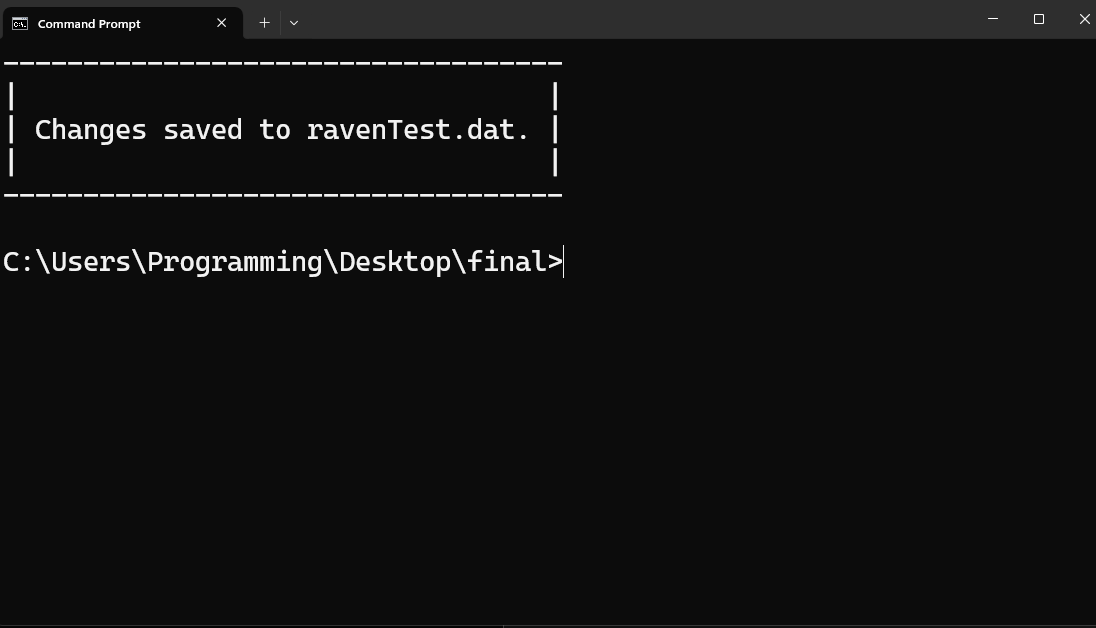


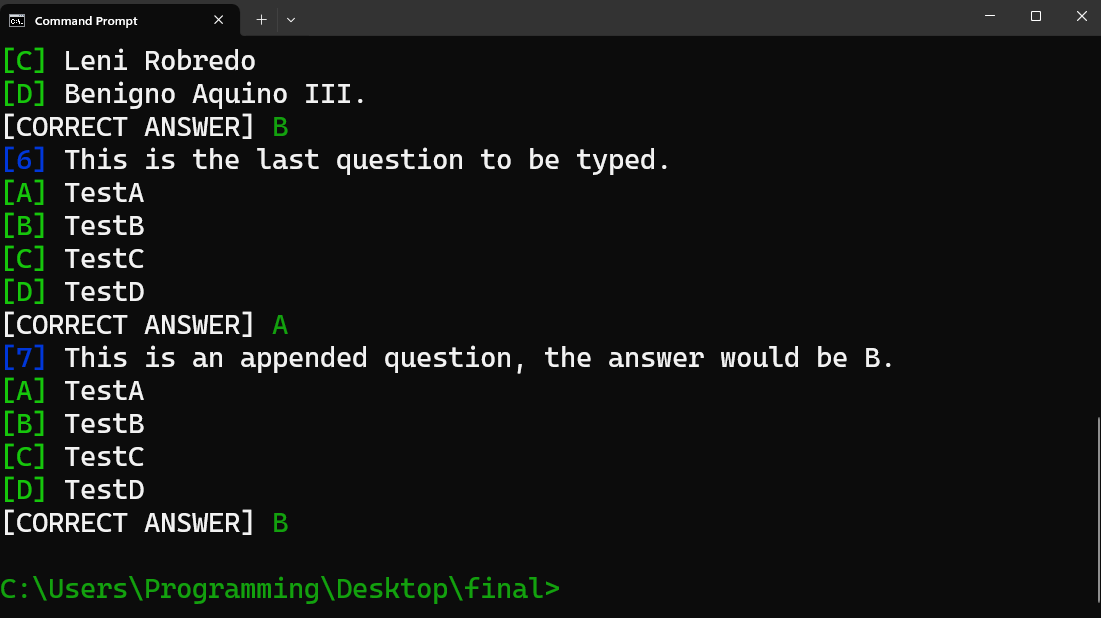
editExam Code Output Samples



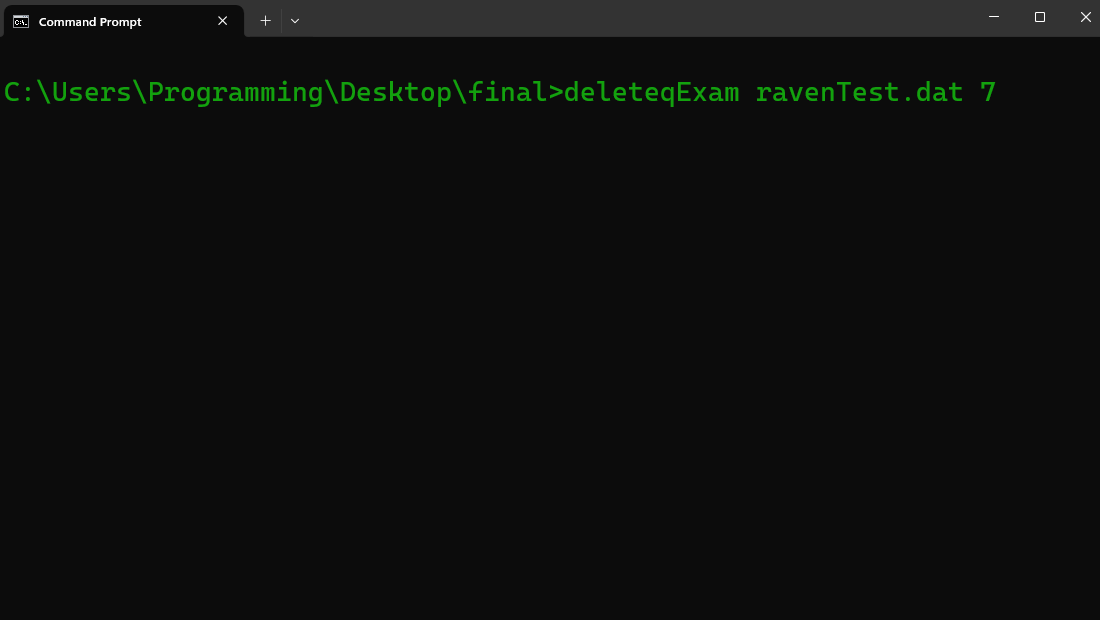
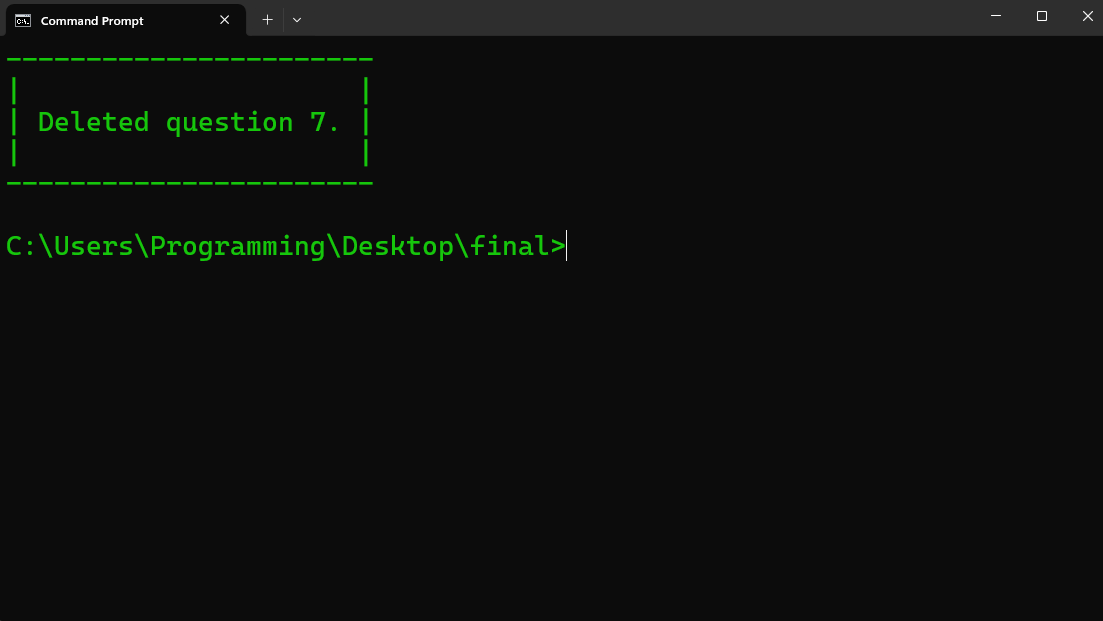
 

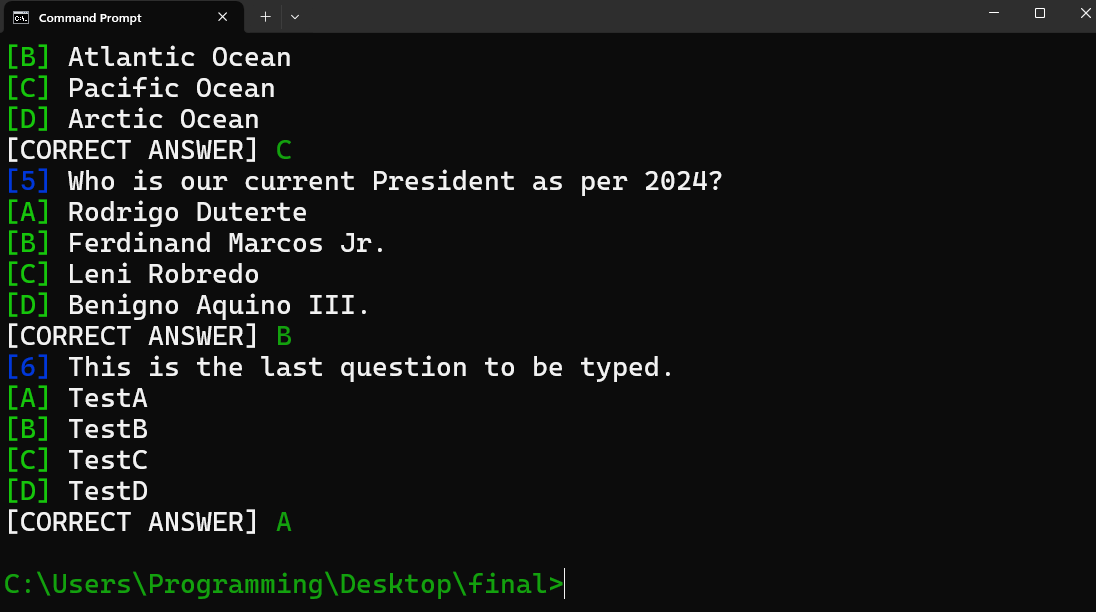
appendExam Code Output Samples





deleteqExam Code Output Samples





doExam Code Output Samples

