

# CS 135 - Computer Science I

## Design Assignment 6 (DA6-10/24)

## Programming Assignment 6 (PA6-10/28)

As specified in your syllabus, you must turn your assignments in by 6:00 pm on the due date specified. If it is turned in late, but prior to 12:00 midnight the day it is due, credit will be reduced by 50% of the earned score. Any laboratories turned in more than 6 hours late will not earn any credit.

### Objectives:

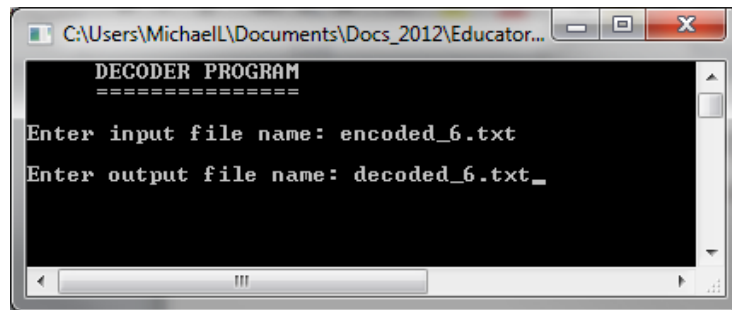
- 1) You will use a design procedure to develop a well-organized and modular program
- 2) You will use reference parameters to develop functions with multiple return quantities
- 3) You will use file input and output (I/O) to access and process data from a file
- 4) You will use simple one-dimensional arrays to upload or read data from a file, store data for processing, and display results of a process
- 5) You will use simple loops to upload or read and process integer values from a file
- 6) You will use a function to conduct some specific file I/O operations

### Tasks:

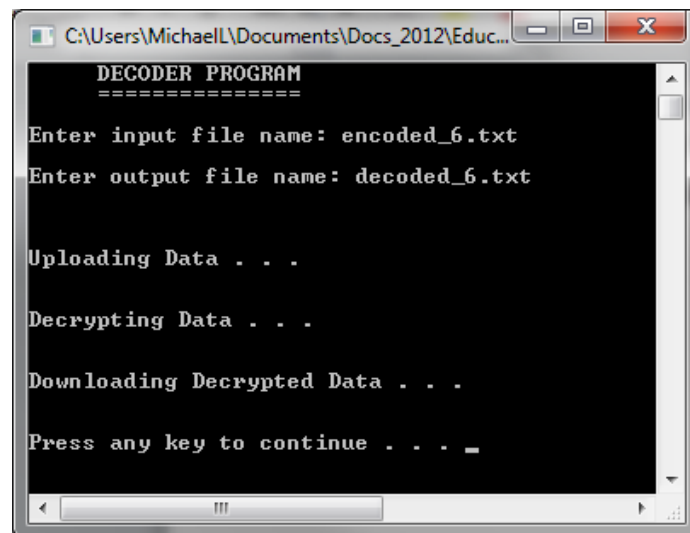
#### *Decoding Encrypted Files, Part Dos*

- 1) Create your program folder as needed. Your program name will be `decoder2.cpp`. You must use `iostream` I/O operations such as `cout` or `cin` for console I/O in this program.
- 2) You will also need to download a series of data files from `encoded_1.txt` up to and including `encoded_6.txt`. These files are the same as last week. Again remember, if you download these as text and paste the data into a file, make sure you add an extra ENTER or two after the last row. This will be necessary for your file input operations.
- 3) You will be redeveloping your text file decoder program to use arrays. As before, the files are comprised of several rows of five integers. The data is encrypted in two ways. First, the only appropriate data is found in even numbers; if the value in the file is an odd number, it must be ignored. Secondly, the data is encoded as the first or most significant two or three digits of a five or six digit number, respectively. The first or most significant two or three digits of each of the five or six digit (even) numbers will be an ASCII character value. Your task will be to capture these values and output them to another file as characters.

- 4) The program will prompt the user for the input file name (e.g., one of the six downloaded files), and then the output file name (e.g., the new file you will be creating with your output). An example of the program operation is shown here:

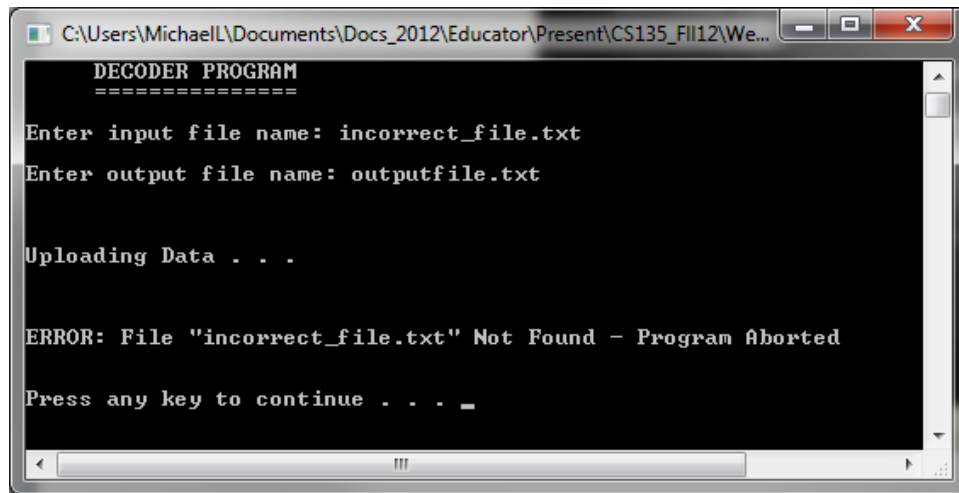


- 5) If the file is found and decrypted, the following screen must be displayed.



- 6) Note the difference in process reports during the program run. Each of these reports must be associated with the three operations specified later in this document. As before, they must only be displayed when the specified action is being conducted; they cannot be displayed together as a group of two or three messages.

- 7) If the input file is not found, the following screen must be displayed and the program must be shut down without conducting any further processing. However, the difference between this result and the previous week's program is that the reading process will be attempted (as shown in the display). However, if the reading process fails, the error message must be displayed and no other processing may occur after that point in the program.



```
DECODER PROGRAM
=====
Enter input file name: incorrect_file.txt
Enter output file name: outputfile.txt

Uploading Data . . .

ERROR: File "incorrect_file.txt" Not Found - Program Aborted

Press any key to continue . . . _
```

- 8) The remaining program specifications are the same as the previous week's assignment, and are provided next for your reference.
- 9) Your program must first upload or read the data from the file. Then it must iterate through the array (instead of the file) checking for odd or even numbers. If your program finds an odd number, this number must be ignored. If the program finds an even number, then the first or most significant two or three digits of the five or six digit number must be acquired, and then stored in the array as an integer. Finally, it must download or write the integer values stored in the array to the file as characters.
- 10) Each data set has a couple of paragraphs in it. So, since you don't want your output to look like one long text line, you must output ten words per line in your output file, and then go to the next line in the output file. This is easily done by identifying and counting the number of spaces in each line as you are reading the characters and outputting them; when you get to the tenth space, output an end of line character.

- 11) For example, if the above two text steps were output to a file in this form, it would look like the following: 1) each line has nine spaces and ends at the tenth space, and 2) when a paragraph end called an end line character (see below) is found, outputting the character to the file will start a new line. However, you will also have to reset your line word counter so that your next line has the appropriate ten words.

Your program must iterate through the encoded file checking for odd or even numbers. If your program finds an odd number, this number must be ignored. If the program finds an even number, then the first or most significant two or three digits of the five or six digit number must be acquired, and then sent to the specified output file as a character (hint: you must force the integer to act like a character to do this).

Since each data set has a couple of paragraphs in it, and you don't want your output to look like one long text line, you must output ten words per line in your output file, and then go to the next line in the output file. This is easily done by identifying and counting the number of spaces in each line as you are reading the characters and outputting them; when you get to the tenth space, output an end of line character.

- 12) Note that a space or an end line marker is a character like any other. However, like all characters in a well-developed program, you must identify the space as a space and the endline character as an endline character. Do not use ASCII numerical values in any of your programs. This can lead to problems with extensibility and maintainability of your programs. More information related to managing the spaces and end lines is provided later in this document.
- 13) When using your `ifstream` object to acquire data with the extraction operator, the input process will ignore all spaces, tabs, and end lines, called whitespace, that are found in the file when it is reading the integers. However the uploaded integers themselves may hold the ASCII value of a space, a tab, or an end line. These are easy to use in this program since you only have to output them to your file as characters - they will do what they are supposed to do.

- 14) As mentioned previously in this document, your process of deciphering the encoded data will be to skip over the unnecessary integers and display the correct integer to the file as a character. However, you must also monitor the input specifically for the following two items: 1) you must watch for a space ( ' ') so that you can count the number of spaces per line as specified previously, and 2) you must watch for an end line value ( '\n' ) which could happen at any time prior to having displayed ten words on a line. Once this end line value is captured and then forwarded to the output file, the output file text will start a new paragraph on the next line. When this happens, you must reset your word counter so that the next line of text output still has no more and no less than ten words. Note however, that the end line character is different between Windows, Mac, and Linux. For this reason, and for just plain good practice reasons, both of these characters should be defined as global constants.
- 15) The program must use at least eight functions, with the following specifications:
- a) one (and only one) function must be called from the main function to conduct the user input operations (e.g., acquiring the two file names) and return these user input data values from that function; the function must prompt the user for each of the two input quantities, acquire the two input quantities, and then return them all at once (i.e., all in one function call)
  - b) one function must be completely and uniquely responsible for opening the file, uploading or reading the data to an array, and closing the file; this function must report when it is reading, or attempting to read the data

- c) one function must be completely and uniquely responsible for decrypting the data loaded to the array. The decryption process must occur “in place”, meaning that the data will be analyzed in each element of the array and then placed back into the same array from which it came in order; you must not use a second array to implement this action. The following is an example using the **encoded\_6.txt** file data.

arr[ 0 ]	arr[ 1 ]	arr[ 2 ]	arr[ 3 ]	arr[ 4 ]	arr[ 5 ]	arr[ 6 ]	arr[ 7 ]	arr[ 8 ]	arr[ 9 ]	arr[ 10 ]
82706	101772	50823	97032	115440	104481	111396	110254	32810	100031	86543

arr[ 0 ]	arr[ 1 ]	arr[ 2 ]	arr[ 3 ]	arr[ 4 ]	arr[ 5 ]	arr[ 6 ]	arr[ 7 ]	arr[ 8 ]	arr[ 9 ]	arr[ 10 ]
82	101	97	115	111	110	32	78	117	109	86543

arr[ 11 ]	arr[ 12 ]	arr[ 13 ]	arr[ 14 ]	arr[ 15 ]	arr[ 16 ]	arr[ 17 ]	arr[ 18 ]	arr[ 19 ]	arr[ 20 ]	arr[ 21 ]
64693	58299	78302	98983	117686	63057	94955	80331	93093	36221	109492

To expand on the graphic shown, the following occurs:

- i) the value at index 0 is even, so it is converted to ASCII and overwrites the same element
- ii) the value at index 1 is even, so it is converted and overwrites the same element,
- iii) the value at index 2 is odd, so it is ignored
- iv) the value at index 3 is even, so it is converted and overwrites the element at index 2
- v) the value at index 4 is even, so it is converted and overwrites the element at index 3
- vi) the value at index 5 is odd, so it is ignored
- vii) the value at index 6 is even, so it is converted and overwrites the element at index 4
- viii) the value at index 7 is even, so it is converted and overwrites the element at index 5
- ix) the value at index 8 is even, so it is converted and overwrites the element at index 6
- x) the values at indices 9, 10, 11, and 12 are odd, so they are ignored
- xi) the value at index 13 is even, so it is converted and overwrites the element at index 7
- xii) the value at index 14 is odd, so it is ignored
- xiii) the value at index 15 is even, so it is converted and overwrites the element at index 8
- xiv) the values at indices 16, 17, 18, 19, and 20 are odd, so they are ignored
- xv) the value at index 21 is even, so it is converted and overwrites the element at index 9

Note the following:

- i) there will always be more available numbers than used (i.e., even) numbers because many are odd; thus you will never overwrite the needed data before you use it; you will also notice that some of the remaining elements will not be overwritten (e.g., the element at index 10) for the same reason
- ii) you will be placing ASCII values in your array with one index while you are acquiring and testing the available values with another index; these indices will proceed for different reasons: 1) the ASCII value index will increase when you store a new found value, and 2) the available data index will increase every time you either skip an odd number or find an even one

- iii) this function must use another function to find the next even number as specified below in item 'd' below; you may also use other functions as needed within this function
  - iv) this function must report when it is conducting the decryption process
  - d) one function must be used to seek and find the next even number in the array, and return it; this function must use a loop so that any number of odd integers can be accommodated. It must also be able to respond to finding the end of the array data during its search; this function must be used inside the function specified in item 'c' above
  - e) one function must be completely and uniquely responsible for downloading or writing the results to the file; this function must report when it is writing data to the file
  - f) there are several other opportunities to create supporting functions for this program, so it should be easy to have more of them; you must however have a minimum of eight functions
- 16) In order to use an array for this activity, you will need to know the number of values in the files. This can be done in one of several ways, some easier than others, but one way might be to create a simple program of your own (i.e., not part of the assignment program) that counts values. Once you know the number of values in the largest file, create your array with several extra elements in order to accommodate the incoming data
- 17) Create a screen shot of each of the two conditions: 1) correctly input file name, processing, and completed program action, and 2) an incorrectly input file name and the error message and program shut down.
- 18) Also provide at least three decrypted files with your assignment folder. This is explained later in this document.

## ***Turning in your Design Assignment:***

### **Information:**

Week: 9

Laboratory:9

Design Assignment: 6

Due Date: 10/24, 6:00 pm

### **To turn in:**

The first five steps of the Six Step Programming Process, including:

1. decoder2\_s1.cpp
2. decoder2\_s2.cpp
3. decoder2\_s3.cpp
4. decoder2\_s4.cpp
5. decoder2\_s5.cpp

***Upload these as separate files, not as a zipped file.*** Note that following the instructions for uploading your work is critical; you will lose points if you do not follow these instructions. Do not upload any files or in any format other than that specified here.

For information on how to turn in Design Assignments, refer to the "How to Turn in Design Assignments" in the "General Course Information" folder



# Turning in your Programming Assignment:

## Information:

Week: 9

Laboratory: 9

Programming Assignment: 6

Due Date: 10/28, 6:00 pm

## To turn in:

1. The Word file containing the following:
  - a. There should be at least two screen shots for the programs as specified in item #18 previously in this document
  - b. Remember to clearly annotate every displayed result
2. The executable file:
  - a. decoder2.exe (decoder2\_s6.exe is also acceptable)
3. The source code file:
  - a. decoder2\_6.cpp
4. At least three decoded files produced from the six encoded input files:
  - a. decoded\_1.txt
  - b. decoded\_2.txt
  - c. decoded\_3.txt
  - d. decoded\_4.txt
  - e. decoded\_5.txt
  - f. decoded\_6.txt

These files must be compressed and uploaded as one zip file. To do this, select all of the required files, right click on them, and select “Send To”, then select “Compressed (zipped) Folder”.

Once the folder is created, it will be placed in the same folder in which you are working. Change the name of the zipped folder to “LastnameFirstname\_PA0X” (where ‘X’ is the number of the Programming Assignment) as shown in the following example: “LeveringtonMichael\_PA06” (no quotes). After you have renamed the zipped folder, double click on it to verify that it has all the files it is supposed to have.

Note that following the instructions for uploading your work is critical; you will lose points if you do not follow these instructions. Do not upload any files or in any format other than that specified here.

For information on how to turn in Programming Assignments, refer to the "How to Upload Your Laboratory Assignments" in the "General Course Information" folder