CS 135 - Computer Science I

Design Assignment 6 (DA6-10/25)

Programming Assignment 7 (PA7-10/29)

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 loops to upload and process integer values from a file
5) You will use a function to conduct some specific file I/O operations

Tasks:

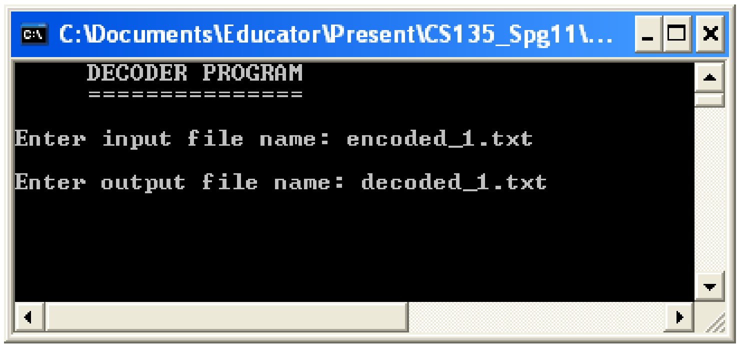
Decoding Encrypted Files

1) Create your program folder as needed. Your program name will be **`decoder.cpp`**. You must use **`iostream`** I/O operations such as **`cout`** or **`cin`** for 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 have information encoded in them that you need to discover as part of your laboratory activity. As a note, 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.

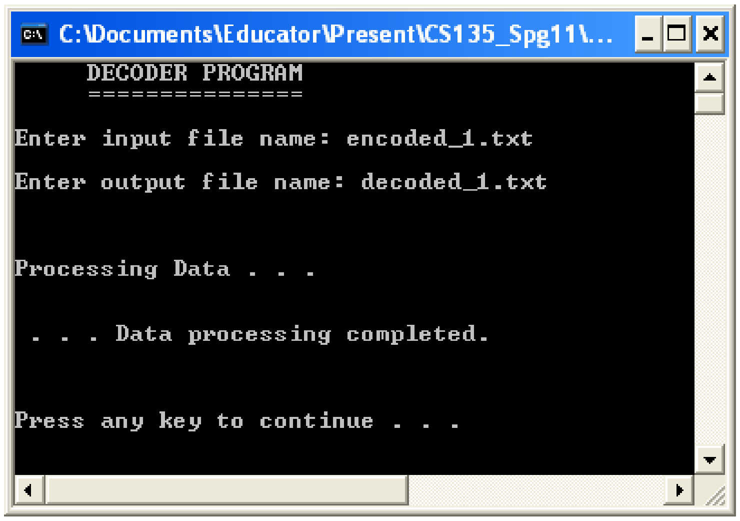
3) You will be creating a program that decodes encrypted text files. The files are comprised of several rows of five integers. The data is encrypted in two ways. First, the only appropriate data is found in even numbers; if the value in the file is an odd number, it must be ignored. Secondly, the data 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 (e.g., one of the six downloaded files), and then the output file (e.g., the new file you will be creating with your output). An example of the program operation is shown here:



```
C:\Documents\Educator\Present\CS135_Spg11\...
     DECODER PROGRAM
     ===============

Enter input file name: encoded_1.txt

Enter output file name: decoded_1.txt
```

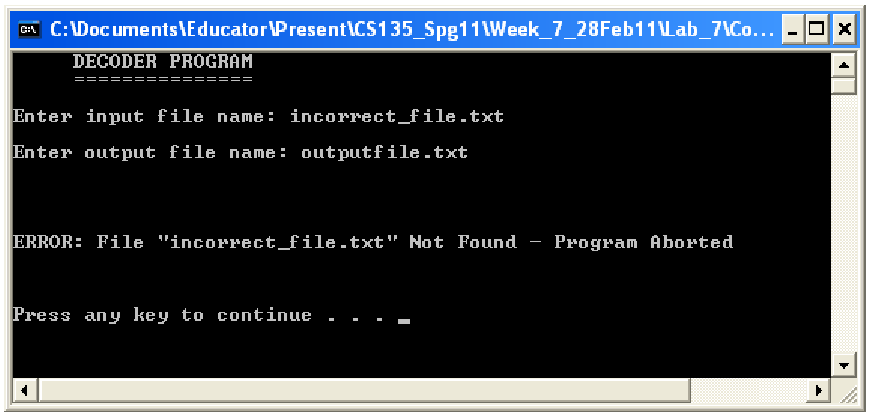
5) If the file is found and decrypted, the following screen must be displayed.



```
C:\Documents\Educator\Present\CS135_Spg11\...
     DECODER PROGRAM
     ===============

Enter input file name: encoded_1.txt

Enter output file name: decoded_1.txt


Processing Data . . .

 . . . Data processing completed.


Press any key to continue . . .
```

6) Note that the "Processing Data" and "Data processing complete" texts seem to be displayed one after the other. However, there is more going on in the program. The "Processing Data" statement is displayed just **before** the files are being opened and the decode process is beginning; the "Data processing completed" statement is displayed **after** the decoding process is completed and the result is downloaded to a file. This protocol must be followed; it would be incorrect to print both statements at the same time.

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.


C:\Documents\Educator\Present\CS135_Spg11\Week_7_28Feb11\Lab_7\Co...

```
     DECODER PROGRAM
     ===============

Enter input file name: incorrect_file.txt

Enter output file name: outputfile.txt



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


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

8) 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).

9) 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 uploading the characters and outputting them; when you get to the tenth space, output an end of line character.

10) 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 uploading the characters and outputting them;
when you get to the tenth space, output an end
of line character.
```

11) 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.

12) You can open the data files in your Dev C++ editor, but be careful not to modify them. The data itself is shown next. This data is from the **`encoded_4.txt`** file. You must read in the numbers from left to right, then continue on the following line, and so on until you reach the end of the file. The first and second numbers are odd, so they would be skipped. However, the third number is even, so its first two most significant digits (i.e., 82) must be stripped from it since it is a five digit number. The third number is also even but for this one, the first three most significant digits (i.e., 101) must be stripped from it since it is a six digit number. As each ASCII value is stripped from the whole number, it can be tested to see if it is a space or an end of line marker, and then it can be sent as a character to the output file. This process must continue through the entire data set in the encoded input file.

```
 34477   77989   82082  101964   97220
115622  111155  111412  116447  110370
 32736   85435   36195   78426  124571
```

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 uploading 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 five 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 used to seek and find the next even number, 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 file during its search

    c) 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 five functions

16) Create a screen shot of each of the two conditions: 1) correctly input file name and completed program action, and 2) an incorrectly input file name and the error message and program shut down.

17) 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: 10/25, 6:00 pm

To turn in:

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

1. decoder_s1.cpp
2. decoder_s2.cpp
3. decoder_s3.cpp
4. decoder_s4.cpp
5. decoder_s5.cpp

Upload these as separate files. 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

Important grading note: Due to the large number of students and the small number of people grading assignments, a fraction of the Design Assignments will be graded each week. If your DA is not graded that week, you will receive full credit; if it is graded, you will receive feedback and the appropriate grade. The random process will ensure that all students will be graded an equal number of times across the semester, and it also provides the possibility of getting graded more than one week in a row. You must do your best on each DA and assume you will be graded each week.

Important DA feedback note: If your DA does not get graded in one particular week, you should do the following:

1. At least look at the provided sample DA when it is uncovered on the Tuesday after the PA is due; make sure your structure, organization, and problem-solving strategies are comparable

2. Stop by one of the Instruction Team offices (i.e., Instructors, TAs, Tutors, etc.) and ask them to help you review your DA so you can be sure to do well when you are graded

Turning in your Programming Assignment:

Information:

Week: 9
Laboratory: 9
Programming Assignment: 7
Due Date: 10/29, 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 #16 previously in this document

    b. Remember to clearly annotate every displayed result

2. The executable file:
    a. decoder.exe (decoder_s6.exe is also acceptable)

3. The source code file:
    a. decoder_6.cpp

4. At least <u>three</u> 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 <u>must</u> 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_PAX” (where ‘X’ is the number of the Programming Assignment) as shown in the following example: “LeveringtonMichael_PA3” (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 Turn in Programming Assignments" in the "General Course Information" folder