Department of Computer Science

CSE1201: Introduction to Programming with C

Group Project Specification

Date due: Wednesday April 30, 2025 at 8pm latest

INSTRUCTIONS

In your main programming assignment for this semester, you will create a message/note storing application in C.

The submission date for this coursework is Wednesday April 30, 2025 no later than 8pm.

Submission:

By the date/time specified above, one member of the group should upload a zip file. The name of the zip file must be in the form:

Group-groupnumber-semester-project.zip

e.g. group-5-semester-project.zip

The zip file must contain your source code, documentation files and any other resource files needed.

Failure to follow the submission specifications by not providing required files or not following the naming conventions and .zip structure specified, may result in penalised marks.

Plagiarism cases will be handled as documented in the University regulations.

Project Concept Specifications

You are to produce a message/note storing application in C that enables users, if they desire, to store notes or messages to a single file that can be later retrieved. The intention is that the application is a single user application. The user can choose to store the notes/messages in their current form or altered.

All the messages/notes will be stored in a single file in a specified location known by the application.

Upon accessing the application, a user should be able to view previously stored notes/messages or enter a new note/message to be stored.

Required features

The application should present the user with 4 main options:

- View all notes/messages
- Search for and view a note/message
- Delete a note/message
- Store a new note/message

Optional features

The application could optionally provide the following options:

- Search for notes/messages by word contained in the notes
- Modify the contents of a previously stored note

View all notes/messages

If the user chooses to view all the notes/messages stored, the application will print the message id, title and partial message/note to screen for each of the messages. The group can decide how many characters to display for the partial message.

Search for and view a note/message

If the user chooses to search for and view a note/message, they will be prompted to enter the message/note id or title after which the application searches for and once found, displays the message id, title and the full message to screen. If the message/note was encrypted, the user should have the option to decrypt the message/note to view it as plain text.

Delete a note/message

If the user chooses to delete a note/message, they will be prompted to enter the message/note id or title after which the application searches for and once found, deletes the note/message from the file.

Store a new a note/message

If the user choses to store a new note/message, they will then be prompted to decide on how the message will be stored

- Store message/note unchanged
- Censor the message/note before storing
- Encrypt the message/note before storing

Store the message/note unchanged

If the user chooses to store the message/note unchanged, the application should accept a title and the contents of the message and store it appropriately in the defined file.

Censor the message/note before storing

If the user chooses to censor the message/note, the application should ask the user to enter a comma separated list of words that should be censored from the message/note.

The program will implement an algorithm that redacts or censors the words provided from the text in the file. It should then accept a title for the message/note and the

contents of the message/note, redact the words if they appear in the message/note and store the result appropriately in the defined file.

For example:
Given the block of text below:

The quick brown fox jumps over the lazy dog

and the redactable set of words:

the, jumps, lazy

the output text stored should be

**** quick brown fox ***** over *** **** dog*

Note

- The number of stars in the redacted text must match the number of letters in the word that has been redacted.
- Capitalization is ignored.
- Only whole words that match one of the redacted words should be redacted.
    - Ignore words that are part of words e.g. *jumpsuit* should not be redacted given the word *jumps*.
    - Ignore hyphenated words and words with apostrophes.

Encrypt the message/note before storing

If the user chooses to encrypt the message/note, the application should ask the user to enter a key to be used for the encryption.

The program will implement a columnar transposition algorithm that encrypts the words provided from the message/note entered. It should accept the contents of the message/file, encrypt it following the guide below and store the result appropriately in the defined file.

Encryption

Input : Attack at dawn

Key = keys

Output : tk-nActwaaa-t-d-

Method

Let's look at how to encrypt the message "Attack at dawn" with the keyword "keys".

- First, we enter the letters into a matrix, row by row, from left to right. The number of columns of the matrix is determined by the key and the size of the matrix depends on the length of the message. Spaces and empty cells in the bottom row of the matrix should be represented by dashes (-).

- Our resulting grid should look like this:

K	E	Y	S
A	t	t	a
c	k	-	a
t	-	d	a
w	n	-	-

- Now that we have constructed the table, the columns are now reordered such that the letters in the keyword are ordered alphabetically:

E	K	S	Y
t	A	a	t
k	c	a	-
-	t	a	d
n	w	-	-

- The ciphertext is now read off column by column, so in our example above, the ciphertext is tk-nActwaaa-t-d-

Decrypt a stored message/note

If the user chooses to decrypt a stored message/note, the application should ask the user to enter a key to be used for the decryption.
The program will implement an algorithm that decrypts the previous columnar transposition. It should then read the contents of the message and decrypt it following the guide below and display the results back to the user.

Decryption

Input : tk-nActwaaa-t-d-
Key = keys
Output : Attack at dawn

Method

- To decipher the text, the algorithm must work out the number of rows by dividing the message length by the key length. Then write the message out column by column, using the alphabetical order of the keys.

- Our resulting grid should look like this:

E	K	S	Y
t	A	a	t
k	c	a	-
-	t	a	d
n	w	-	-

- Then re-order the columns by reforming the key word.

K	E	Y	S
A	t	t	a
c	k	-	a
t	-	d	a
w	n	-	-

- After reforming the keyword, the decrypted text is now read off row by row, so in our example above, the decrypted text is Attack at dawn. The dashes are converted to spaces.

Search for a note/message using a word/phrase from the message (Optional)

If the user chooses to search for and view a note/message, they can be prompted to enter a word/phrase from the message/note id or title after which the application searches for messages matching this criteria and once found, displays the message id, title and the full message to screen. If the message/note was encrypted, the user should have the option to decrypt the message/note to view it as plain text.

Modify an existing message (Optional)

If the user chooses to modify an existing note/message, they will be prompted to enter the message/note id or title after which the application searches for and once found, allows the user to modify the text stored. If the message/note was encrypted, the message/note should be decrypted before it is modified.

The following functions must be included in your application:

- ***char * read_string(char *filename)*** - a function which takes in a filename and returns the contents as a string.

- ***void write_string(char *filename, char* message)*** - a function which takes in a filename and the contents as a string and writes the contents to the file.

- ***char * redact(char *message, char *list)*** - a function which takes in a message and redacts the words from the list provided. The redacted message should be returned.

- ***char * encrypt_columnar(char *message, char *key)*** - a function which takes in a message and encrypts it using the key provided. The encrypted message should be returned.

- **char * decrypt_columnar(char *message, char *key)** - a function which takes in message and decrypts it using the key provided. The decrypted message should be returned.

Markscheme:

Criteria	Max Score	Description
Required Functionality (80)		
Basic Functionality	2	Program runs without errors, accepts user's input and presents the options, accepts choice and presents the options for completing individual choice.
Read from file	4	Successfully reads the contents of the file
Write to file	6	Successfully stores new messages to the file
Displays the contents of the file	10	Successfully displays the contents of a file when the user choses to view all or one at a time (search)
Search for a message	8	Successfully searches the messages to find the indicated messages, reports error if doesn't exist/found.
Censors message contents	10	Successfully censors the contents of the file
Encrypts message contents	20	Successfully encrypts the contents of the file
Decrypts message contents	20	Successfully decrypts the contents of the file
Code Quality (15)		
Good code practises	10	Appropriate use of methods for modularity. Good code technique and good management of memory.
Formatting and commenting	5	Indentation, whitespace, sensible comments throughout the submission
Documentation (15)		
User documentation	5	A manual that describes how to start and use the application for a user
Technical documentation	10	A document that describes the code and its structure to a technical audience who would possibly want to maintain or modify the code.
Optional Functionality (15)		
Both optional features	15	The inclusion of both optional features working as expected.
One optional feature	8	Inclusion of at one optional feature working as expected

Total Score Calculation:

A group that undertakes the required functionality and requisite documentation can only gain a maximum of 110/125 which equates to a score of about 88% which would be enough for an A for this assessment.

A group that undertakes the required and optional functionality can gain a maximum of 125/125.