EECS2031 Winter 2018

Software Tools

Assignment 3 (15%): C Programming

Due Date: 11:59pm on Thu, Apr 5, 2018

Objective

In this assignment, you will be writing two C programs. The first program (**encrypt.c**) implements a Caesar cipher, used to encrypt text messages in files. The second program (**team.c**) maintains information for a soccer team. The data in your team will be stored in memory with the use of a linked list, with list nodes representing players.

Important Notes:

- You must use the *submit* command to electronically submit your solution by the due date.
- All programs are to be written using *C* and compiled by *gcc*.
- Your programs should be tested on the *EECS labs* before being submitted. Failure to test your programs on *EECS labs* will result in a mark of 0 being assigned.
- To get full marks, your code must be well-documented.

What To Submit

When you have completed the assignment, move or copy your four C programs in a directory (e.g., assignment3), and use the following command to electronically submit your files within that directory:

```
% submit 2031M a3 encrypt.c team.c
```

You can also submit the files individually after you complete each part of the assignment– simply execute the *submit* command and give the filename that you wish to submit. You may submit your solutions as many times as you wish prior to the submission deadline. Make sure you name your files **exactly** as stated (including lower/upper case letters). Failure to do so will result in a mark of 0 being assigned. You may check the status of your submission using the command:

```
% submit -l 2031M a3
```

A. Caesar's Cipher for Files (20%)

Write a C program **"encrypt.c"** that encrypts a message using one of the oldest known encryption techniques, called Caesar cipher, attributed to Julius Caesar. It involves replacing each letter in a message with another letter that is a fixed number of positions later in the alphabet (shift). If the replacement would go past the letter *Z*, the cipher "wraps around" to the beginning of the alphabet. For example, if each letter is replaced by the letter two positions after it (shift by 2), then *A* would be replaced by *C*, *Y* would be replaced by *A*, and *Z* would be replaced by *B*. The program should get two arguments:

(a) the name of a file containing the message to be encrypted and
(b) the shift amount as a valid integer from 1 to 25 (inclusive).

Here's an example of a call to your program:

```
$ ./encrypt message.txt 3
```

The program then writes the encrypted message to a new file with the same name but an added extension of *.enc*. In the example above, the original file name is *message.txt*, so the encrypted message will be stored in a file named *message.txt.enc*. You may assume that:

- There is no limit on the size of the file to be encrypted or on the length of each line in the file.
- Characters other than letters should be left unchanged.
- Lower-case letters remain lower-case and upper-case letters remain upper-case.

Notice that the program can also be used to decrypt a message if the user knows the original key by providing the encrypted message and using as shift amount 26 minus the original key. As an example:

```
$ ./encrypt message.txt.enc 23
```

Hint: You may wish to use operations on characters to handle the "wrap around" problem and to calculate the encrypted version of a lower-case or upper-case letter.

B. Linked Lists (80%)

Write a C program **"team.c"** that maintains information for a soccer team. The program will allow you to add and delete players from your team, to search your team for players by name or by value they are worth, and to print out part or all of the team.

The data in your team will be stored in memory with the use of a linked list, with list nodes representing players. Each node will contain members for storing a player's family name (`char *`) and first name (`char *`), their position (`char`) and their value (`int`). There are four possible positions, each of which is identified by the first character in the words (*G*)oalkeeper, (*D*)efender, (*M*)idfielder and (*S*)triker. Your linked list must be kept in a special order, with all the goalkeepers first, then the defenders, then the midfielders, and finally the strikers. If there is more than one player in the same position, then the players should be kept in order of their insertion (e.g., the last defender in the team, should be the defender most recently inserted into the list; the first striker in the team should be the striker that was inserted first into the list, and so on). You may assume that no two players that assume the same position in your team have the same family name.

Your program should be menu driven, with the user being offered a choice of the six commands described below:

- *Insert a new player into the team*. The program should prompt the user for a new family name and first name, a position and a value. This information should be placed in a new node that has been created using the `malloc` function. And then the node should be inserted at the appropriate position in the linked list that stores the team data. Don't forget that the team must be stored in a special order, by considering the player's position first, and then (if needed) the order of insertion. If a node with the given family name is already in the team, an error message should be produced and the new node should not be inserted into the linked list.
- *Delete a player from the team*. The program should prompt the user for the family name of the player to be deleted and then delete the node containing that family name from the linked list that stores the team. If no player with the given family name is found in the team, an error message should be produced.
- *Search for a player using an input **family name***. The program should print the family name, first name, position and value of the player, with each piece of information on a separate line. If no player with the given family name is found in the team, an error message should be produced.
- *Search for players in the team that are worth less than or equal to an input **value***. The program should print the family name, first name, position and value of any player that is worth less than or equal to an input value, with each piece of player information on a separate line. A blank line should be printed between each player (if more than one is found). If no player in the team is worth less than or equal to the given value, an error message should be produced.
- *Print the team, following a special order*. Print the family name, first name, position and value of each player, with each piece of information on a separate line. A blank line should be printed between each player. The special order assumes that the goalkeepers appear first, then the

defenders, then the midfielders, and finally the strikers. If there are more than one players of the same position, then they should be printed in order of their insertion.

- *Quit the program*. When the program is given the quit command, it should delete all nodes in the linked list by using calls to the `free` function. It should then try to print the linked list.

To assist you in the production of your program, we will provide you a file that contains a skeleton of a complete program. You can download the skeleton program here:

https://www.eecs.yorku.ca/~papaggel/courses/eecs2031/docs/assignments/team.c

The skeleton program includes all of the C statements required to implement the menu driven parts of the program. It also includes a few helpful functions for reading data and printing messages. All you need to do is implement the instructions for working with the linked list that stores the team.

It is recommended that you take the following steps:

- Read the whole skeleton program carefully. Take note of the provided functions for reading strings, printing information about players, and for printing error messages. Use of these functions will make it easier for you to satisfy the tester and marker programs.
- Figure out how to express the required player information in a node. Add a `struct` definition for a linked list node to the program.
- Add the function for inserting a node into the linked list. Your function will need to read the family name, first name, position and value for the player. Test your program by trying to insert nodes into the linked list. Try to insert nodes with both new and duplicate family names, and with different positions.
- Add a function for printing the linked list. Test your program by inserting entries into the linked list and then printing them out. Are the entries in the correct order?
- Add a function that searches the linked list for a player with the given **family name** and then either prints the appropriate entry or, if a player with the given family name is not found, prints an error message.
- Add a function that searches the linked list for players that are worth less than or equal to a given **value** and then either prints the appropriate entries or, if a node is not found, prints an error message.
- Add the statements that need to be executed when the Quit command is entered. These statements should delete the linked list by using calls to the free function. To check your work, print the linked list after all of the elements have been deleted.
- Add a function for deleting a player. It will need to search the linked list for a given family name, delete the appropriate node from the linked list and then use the free function to release the memory used to store the node. If the given family name is not found in the team, print an error message.

It is recommended that you complete and test each step before moving on to the next one. This way, if your program no longer works, you will know which statements are causing the error.

Appendix. Sample Output of Program Execution

Here is a sample output from an execution of the program that you are to prepare. There is one space after each colon (:).

```
Personal Team Maintenance Program.

Commands are I (insert), D (delete), S (search by name),
  V (search by value), P (print), Q (quit).

Command?: I
  family name: Messi
  first name: Lionel
  position: M
  value: 50000

Command?: P

My Team:

Messi
Lionel
M
50000

Command?: I
  family name: Casillas
  first name: Iker
  position: G
  value: 40000

Command?: P

My Team:

Casillas
Iker
G
40000

Messi
Lionel
M
50000

Command?: I
  family name: Henry
  first name: Thierry
  position: S
  value: 35000

Command?: P

My Team:
```

```
Casillas
Iker
G
40000

Messi
Lionel
M
50000

Henry
Thierry
S
35000

Command?: I
  family name: Cannavaro
  first name: Fabio
  position: D
  value: 20000

Command?: P

My Team:

Casillas
Iker
G
40000

Cannavaro
Fabio
D
20000

Messi
Lionel
M
50000

Henry
Thierry
S
35000

Command?: I
  family name: Nedved
  first name: Pavel
  position: M
  value: 32000

Command?: P

My Team:
```

```
Casillas
Iker
G
40000

Cannavaro
Fabio
D
20000

Messi
Lionel
M
50000

Nedved
Pavel
M
32000

Henry
Thierry
S
35000

Command?: Z

Invalid command. Commands are I (insert), D (delete), S (search by
name),  V (search by value), P (print), Q (quit).

Command?: S

Enter family name to search for: Zidane

The player with family name <Zidane> is not in the team.

Command?: S

Enter family name to search for: Messi

The player with family name <Messi> was found in the team.

Messi
Lionel
M
50000

Command?: V

Enter value: 30000

Cannavaro
Fabio
D
```

```
20000

Command?: V

Enter value: 10000

No player(s) in the team is worth less than or equal to <10000>.

Command?: D

Enter family name for entry to delete: Cannavaro

Deleting player with family name <Cannavaro> from the team.

Command?: P

My Team:

Casillas
Iker
G
40000

Messi
Lionel
M
50000

Nedved
Pavel
M
32000

Henry
Thierry
S
35000

Command?: Q

The team is empty.
```