VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

PROGRAMMING FUNDAMENTALS - CO1027

ASSIGNMENT 1

THIRD MONGOL INVASIONS OF VIETNAM

HO CHI MINH CITY, 05/2021

ASSIGNMENT'S SPECIFICATION

Version 1.0

1 Assignment's outcome

After completing this assignment, students review and make good use of:

- Object Oriented Programming (OOP)
- Linked list
- Dynamic array

2 Introduction

Twice invaded Dai Viet have failed, as the Yuan Dynasty emperor angry so determined to beat Dai Vietnam in retaliation for the third time. Kublai Khan stopped attacking Japan and concentrated on attacking Dai Viet. The war took place on Dai Viet territory from December 1287 to the end of April 1288.

In this war, the Yuan Dynasty mobilized more than 30,000 troops and many famous generals under the command of Toghon. According to the history of the Yuan Dynasty, in addition to mobilizing soldiers in the second invasion escaping to China, the Yuan also mobilized more troops from Mongolia and Han, the old Southern Song army surrendered to the Yuan, ... In terms of navy, the Yuan army also mobilized many food transport boats led by Truong Van Ho to accompany Baghatur. Kublai Khan also instructed Toghon not to consider Giao Chi (Jiaozhi in Chinese) as a small country that he despised.

Faced with the danger of being invaded, King Tran urgently prepared to fight the enemy. The Dai Viet side conducted a general mobilization. General commanding the entire army was Hung Dao Vuong Tran Quoc Tuan. Tran Nhan Tong pardoned the world, except those who had surrendered to the Yuan army were not granted amnesty. Many prisoners were volunteered to join the army to show gratitude. Tran Quoc Tuan, with the combat experience gained after defeating the Yuan army 2 years ago, after analyzing the situation of the Yuan army, confidently told King Tran: *"This year's enemy is easy to break"* .

At the end of December 1287, about 30,000 Yuan troops entered our country. The infantry, led by Toghon, crossed the border to attack Lang Son, Bac Giang and then pulled back to Van Kiep. The marine commanded by O Ma Nhi followed the sea route to the Bach Dang River,

then headed to Van Kiep. At Van Don, Tran Khanh Du commanded an ambush, when Truong Van Ho's supply boats arrived, the Tran army fought fiercely. Most of the enemy's ships were sunk, the rest was captured by our troops. At the end of January 1288, Toghon pulled his troops into the empty Thang Long citadel. After the battle of Van Don, the situation of the Yuan army became more and more difficult, many key places were attacked by the Tran army, food was increasingly depleted, and the enemy in Thang Long faced an isolated situation. Toghon decided to withdraw his troops to Van Kiep and from here withdrew his troops to the country by both water and land way.

The Tran Dynasty launched counterattacks on both water and land battlefields. Our troops arranged and ambushed at Bach Dang river. In April 1288, Baghatur's fleet entered the battlefield of piles on the Bach Dang River, arranged by the Tran army in advance. Fierce fighting took place, O Ma Nhi was captured alive. On land, Toghon led troops from Van Kiep in the direction of Lang Son to retreat to China, and was repeatedly intercepted by our troops.

Originally disastrous military defeat, crushed dreams of invading Dai Viet, three times the resistance against the Mongol invaders ended.

3 Input data

3.1 Description of the files in the assignment

In this assignment, student was asked to complete 5 missions, from 0 to 4

- **main.cpp**: is the file containing function main, where the program starts. Function main will call in turn the functions to solve the missions, these functions will be located in the corresponding header files. For example, function *solveTask0* will be in file *task0.h*.
- **dataStructure.h**: is the file containing the definition of the structs used in this assignment.
  - struct Soldier: contains information about a soldier, including HP (hit point), isSpecial (true if special soldier and false if vice versa), ID (ID of that soldier).
  - struct SoldierNode: contains node information for use in a Singly Linked List (SLL) **SLinkedList**.
  - struct SLinkedList: contains information of a SLL, where each node stores information about a soldier.
  - struct Array: contains the information of the dynamic array, where each element in the array stores the information of a soldier.

- **SLLDataController.h** và **ArrayDataController.h**: are files that contain functions to operate on SLL and dynamic array, respectively.
- **thirdBattle.h**: are files containing functions for mission 1 through 4.
- **task<i>.h**: where, i from 0 to 4, is the file containing the function to handle reading the testcase and running the function related to the mission, outputting the results.
- **testcase<i>.txt**: where, i from 0 to 4, are files containing testcases. Note, in these tesecase files, there are NO spaces at the end of each line.

In this assignment, student was asked to implement all functions in files: **SLLDataController.h**, **ArrayDataController.h**, **thirdBattle.h**, and also just submit these files. In addition, students are not allowed to include any libraries, however, can write additional functions in the three files mentioned above, where to write additional functions are clearly noted in each file.

Description of input data for each testcase will be well explained in each mission.

4 Mission

Students are asked to build a fictional program in C++ to simulate the process of preparing soldiers, through the missions described below. Note, a mission can have many functions to implement.

4.1 Mission 0

In this mission, student has to implement functions in 2 files: **SLLDataController.h** and **ArrayDataController.h** to get the dynamic array and SLL management tools needed for the tasks behind.

Function description in file **SLLDataController.h**, note, the last element of the linked list must point to NULL:

- **print**:
  - Parameter: linked list *list*
  - The function will print to the screen the elements in the passed linked list
  - Symbols in testcase: PRINT
- **insertAt**:

  - Parameters: linked list *list*, information about a soldier *element*, index to insert *pos*
  - Function will insert to *list* a soldier. After inserting, a new soldier will be in the position *pos*. The function returns 1 if the insert successfully and 0 if vice versa.
  - Symbols in testcase: INSERTAT[space]<pos>[space]<HP>[space]<isSpecial>[space]<ID>

- **removeAt**:
  - Parameters: linked list *list*, index to remove *pos*
  - Function will remove from *list* a soldier at index *pos*. The function returns 1 if remove successfully and 0 if vice versa.
  - Symbols in testcase: REMOVEAT[space]<pos>

- **removeFirstItemWithHP**:
  - Parameters: linked list *list*, HP of soldier to remove *HP*
  - Function will remove from *list* a first soldier having HP equal to *HP* of parameter. The function returns 1 if remove successfully and 0 if vice versa.
  - Symbols in testcase: REMOVEHP[space]<HP>

- **indexOf**:
  - Parameters: linked list *list*, information of a soldier need to find *soldier*
  - Function will find in *list* the index of the first soldier having three information ID, HP and isSpecial that match the three information stored in *soldier* of the parameter. The function returns -1 if no soldiers are found.
  - Symbols in testcase: INDEX[space]<HP>[space]<isSpecial>[space]<ID>

- **size**:
  - Parameter: linked list *list*
  - The function returns the number of soldiers in the linked list
  - Symbols in testcase: SIZE

- **empty**:
  - Parameters: linked list *list*
  - The function returns true if the linked list is empty and false if vice versa
  - Symbols in testcase: EMPTY

- **clear**:
  - Parameters: linked list *list*
  - The function will remove all elements in the list
  - Symbols in testcase: CLEAR

- **getIDAt**:

  - Parameters: linked list *list*, index *pos*
  - Returns the ID of the soldier at position *pos*, returns -1 if the position is not found in the linked list.
  - Symbols in testcase: GETID[space]<pos>

- **getHPAt**:
  - Parameters: linked list *list*, index *pos*
  - Returns the HP of the soldier at position *pos*, returns -1 if the position is not found in the linked list.
  - Symbols in testcase: GETHP[space]<pos>

- **setHPAt**:
  - Parameters: linked list *list*, hit point *HP*, index *pos*
  - Set the soldier's hit point at the position *pos* with *HP*. Returns 1 if sets successfully and 0 if vice versa
  - Symbols in testcase: SETHP[space]<pos>[space]<HP>

- **contains**:
  - Parameters: linked list *list*, information of a soldier need to find *soldier*
  - The function returns 1 if the soldier *soldier* can be found in the linked list and 0 if vice versa.
  - Symbols in testcase: CONTAINS[space]<HP>[space]<isSpecial>[space]<ID>

Function description in file **ArrayDataController.h**:

- **print**:
  - Parameters: array *array*
  - The function will print to the screen the elements in the passed array
  - Symbols in testcase: PRINT

- **initArray**:
  - Parameters: array *array*, maximum capacity *cap*
  - The function will initialize the dynamic array with the number of elements *cap*
  - Symbols in testcase: INIT[space]<cap>

- **insertAt**:
  - Parameters: array *array*, information about a soldier *element*, index to insert *pos*
  - Function will insert to *array* a soldier. After inserting, a new soldier will be in the position *pos*. The function returns 1 if the insert successfully and 0 if vice versa

  - Symbols in testcase: INSERTAT[space]<pos>[space]<HP>[space]<isSpecial>[space]<ID>

- **removeAt**:
  - Parameters: array *array*, index to remove *pos*
  - Function will remove from *array* a soldier at index *pos*. The function returns 1 if remove successfully and 0 if vice versa.
  - Symbols in testcase: REMOVEAT[space]<pos>

- **removeFirstItemWithHP**:
  - Parameters: array *array*, HP of soldier to remove *HP*
  - Function will remove from *array* a first soldier having HP equal to *HP* of parameter.The function returns 1 if remove successfully and 0 if vice versa.
  - Symbols in testcase: REMOVEHP[space]<HP>

- **indexOf**:
  - Parameters: array *array*, information of a soldier need to find *soldier*
  - Function will find in *array* he index of the first soldier having three information ID, HPand isSpecial that match the three information stored in *soldier* of the parameter.The function returns -1 if no soldiers are found.
  - Symbols in testcase: INDEX[space]<HP>[space]<isSpecial>[space]<ID>

- **size**:
  - Parameters: array *array*
  - eturns the number of soldiers in the array
  - Symbols in testcase: SIZE

- **empty**:
  - Parameters: array *array*
  - The function returns true if the array is empty and false if vice versa
  - Symbols in testcase: EMPTY

- **clear**:
  - Parameters: array *array*
  - The function will delete all the elements in array, and make the pointer pointing to the array in the Array structure point to NULL.
  - Symbols in testcase: CLEAR

- **getIDAt**:
  - Parameters: array *array*, index *pos*

  - Returns the ID of the soldier at position *pos*, returns -1 if the position is not found in the array.
  - Symbols in testcase: GETID[space]<pos>

- **getHPAt**:
  - Parameters: array *array*, index *pos*
  - Returns the HP of the soldier at position *pos*, returns -1 if the position is not found in the array.
  - Symbols in testcase: GETHP[space]<pos>

- **setHPAt**:
  - Parameters: array *array*, hit point *HP*, index *pos*
  - Set the soldier's hit point at the position *pos* with *HP*. Returns 1 if sets successfully and 0 if vice versa
  - Symbols in testcase: SETHP[space]<pos>[space]<HP>

- **contains**:
  - Parameters: array *array*, information of a soldier need to find *soldier*
  - The function returns 1 if the soldier *soldier* can be found in the array and 0 if vice versa.
  - Symbols in testcase: CONTAINS[space]<HP>[space]<isSpecial>[space]<ID>

In this mission, testcase has 1 + n line, with the first line containing the number of commands (N), and the next N line is the command to manipulate. Note, at the begin of each command will have characters to specify which structure was manipulated, "A" with array and "L" with the linked list.

Example 1:

3
A INIT 10
L PRINT
A INSERTAT 3 2 1 H

Testcase has 3 commands, the first command will initialize array with capacity 10, the second command will print the list, and the 3rd command will insert a soldier to array at position 3, with HP, isSpecial ID are 2, 1 and H respectively.

4.2 Mission 1

Students can find functions related to this mission in file **thirdBatttle.h**. In this mission, we build a data structure to manage soldiers from array.

This array will have Last In First Out properties (LIFO) - meaning that the last element added to array will be the first element to leave array. In this mission, students need to implement three functions:

- **push**:
  - Parameters: array *array*, information of a soldier *soldier*
  - The function will insert a soldier to array. The function returns 1 if the insert successfully and 0 if vice versa
  - Symbols in testcase: PUSH[space]<HP>[space]<isSpecial>[space]<ID>
- **pop**:
  - Parameters: array *array*
  - Remove a soldier from array, return 1 if removes success and 0 if vice versa
  - Symbols in testcase: POP
- **top**:
  - Parameters: array *array*
  - The function will return the soldier with the time in the array is the shortest. The default return value was written in the function if the operation cannot be performed.
  - Symbols in testcase: TOP

Note, the function **push** and **pop** must follow the LIFO properties mentioned above. In this task, testcase has 1 + n line, with the first line containing the number of commands (N), and the next N line is the command to manipulate.

Example 2:
3
PUSH 10 0 H
TOP
POP

Testcase has 3 command. The first command will insert soldiers with ID = "H", HP = 10 and isSPecial = 0 into array. The second command will get the soldier information

from array, and the third command will remove a soldier.

4.3 Mission 2

Students can find functions related to this mission in file **thirdBatttle.h**. In this mission, we build a data structure to manage soldiers from linked list.

This linked list will have First In First Out (FIFO) properties - meaning that the first element added to list will be the first element to leave list. In this mission, students need to implement three functions:

- **enqueue**:
  - Parameters: linked list *list*, information of a soldier *soldier*
  - The function will insert a soldier list. The function returns 1 if the insert successfully and 0 if vice versa
  - Symbols in testcase: EN[space]<HP>[space]<isSpecial>[space]<ID>
- **dequeue**:
  - Parameters: linked list *list*
  - Remove a soldier from list, return 1 if removes success and 0 if vice versa
  - Symbols in testcase: DE
- **front**:
  - Parameters: linked list *list*
  - The function will return the soldier with the time in the array is the longest. The default return value was written in the function if the operation cannot be performed.
  - Symbols in testcase: FRONT

Note, the function **enqueue** and **dequeue** must follow the FIFO properties mentioned above. In this task, testcase has 1 + n line, with the first line containing the number of commands (N), and the next N line is the command to manipulate.

Example 3:

```
3
EN 10 0 H
FRONT
DE
```

Testcase has 3 command. The first command will insert soldiers with ID = "H", HP = 10and isSpecial = 0 into list. The second command will get the soldier information from list, and the third command will remove a soldier.

4.4 Mission 3

Students can find functions related to this mission in file **thirdBattle.h**. In this mission, we reverse the data in a linked list through the function **reverse**.

Testcase has N lines containing information of N soldiers to insert to linked list. Insert function was implement in the file **task3.h**. Student was asked to implement function **reverse** to reverse this list.

Example 4:

3
12 1 K
23 0 F
2 1 G

Testcase contains 3 soldiers with information (HP, isSpecial, ID) in order (12, 1, K), (23, 0, F), (2, 1, G).
When three soldiers were read on, *list* is stored as follows (if students implement exactly function **insertAt** in the task 0): $(2, 1, G) \rightarrow (23, 0, F) \rightarrow (12, 1, K) \rightarrow$ NULL

Function **reverse** will reverse this list and *list* becomes: NULL $\leftarrow (2, 1, G) \leftarrow (23, 0, F) \leftarrow (12, 1, K)$

4.5 Mission 4

Students can find functions related to this mission in file **thirdBatttle.h**. In this mission, we merge two lists with order of soldiers into a new list with order.

The soldiers are arranged in the order of priority as follows in the list:

- The most priority: between the two soldiers, who has lower HP will be in front (smaller

index).

- The second priority: If the two soldiers have the same HP, the soldiers with isSPecial flag is false will be in front (smaller index).
- The third priority: If the two soldiers have the same HP and isSpecial, the soldiers have a smaller ID according to the alphabet that will be in front (ID only has 1 character).

Testcase contains information of soldiers and is added to *list1* and *list2*, with 1 + n + 1 + m line, corresponding to the soldiers of *list1* and soldiers of *list2*. Insert function was implement in the file **Task4.h**. Student was asked to implement function **merge** to merge these two lists with order into a new list with order as above description. Note, two initial child lists also comply with the above order.

Example 5:

3
2 0 A
2 0 B
2 1 A
2
1 0 D
2 1 E

Testcase contains 5 soldiers with information (HP, isSpecial, ID)
list1 is stored as follows (if students implement exactly function **insertAt** in the task 0): $(2, 0, A) \rightarrow (2, 0, B) \rightarrow (2, 1, A) \rightarrow$ NULL
list2 is stored as follows (if students implement exactly function **insertAt** in the task 0): $(1, 0, D) \rightarrow (2, 1, E) \rightarrow$ NULL

After merging, the new list will be: $(1, 0, D) \rightarrow (2, 0, A) \rightarrow (2, 0, B) \rightarrow (2, 1, A) \rightarrow (2, 1, E) \rightarrow$ NULL

5 Submission

Students submit 3 files: **SLLDataController.h**, **ArrayDataController.h**, and **thirdBattle.h** on site `Ky thuat lap trinh (CO1027)_HK202` on BKeL.

Deadlines for submission are announced at the submission site above. By the deadline for submission, the link will be locked automatically, so students will not be able to submit them

late. To avoid possible risks at the time of submission, students MUST submit their assignment at least **one hour** before the deadline.

6 Handling fraud

Assignment must be done BY YOURSELF. Students will be considered fraudulent if:

- There is an unusual similarity between the source code of the submissions. In this case, ALL submissions are considered fraudulent. Therefore, students must protect the source code of their assignments.
- Students do not understand the source code written by themselves, except for the parts of the code provided in the initialization program. Students can consult from any source, but make sure they understand the meaning of all the lines they write. In the case of not understanding the source code of the place they refer, students are especially warned NOT to use this source code; instead use what has been learned to write programs.
- Mistakenly submit another student's assignment on your personal account.

In the case of cheating, students will get a 0 for the entire subject (not just the assignment).

DO NOT ACCEPT ANY INTERPRETATION AND NO EXCEPTION!

After each major assignment has been submitted, a number of students will be called for random interviews to prove that the assignment has been done by themselves.

7 Notes on assignment

- Students can change order of the functions, write auxiliary functions, add prototype of the functions at the beginning of the submission files. However, do not change the name of the given functions.
- Capacity means the maximum size the array can reach, the size means the number of elements at a particular instance of the array.
- Whenever inserting into the array or linked list, students must ensure that the elements are contiguous (there are no NULL between 2 elements in the linked list, and no trash between 2 elements in the array).
- To avoid unexpected error, student should increase capacity by following rule: new capacity = capacity * 1.5, and do not allocate too many elements in the array.

8 Change from previous version

References

[1] Le van Sieu (2004), Viet nam van minh su cuong, Publishers Thanh nien.

[2] Ha Van Tam va Pham Thi Tam (1972), Cuoc khang chien chong quan xam luoc Nguyen Mong the ky XIII, Publishers Quan doi Nhan dan, reprint in 2003.

——————————END——————————