Assignment #3, Object Oriented Programming. 2021 Spring Semester

Due date: 2021/05/28,23:59:59

***If dynamic allocation is not deallocated, you will have -0.2 point penalty for each task**

***Handle exceptions for possible exceptions**

---

1. Implement a word chain program using linked list. The program receives a word from user and check it can chain or not. It can connect if the following 2 conditions are met.

- ✓ Is it a word that has not been entered before?
- ✓ Is the word that starts with the last letter of previous entered word (case insensitive)?

Example

<case 1>

```
CMD(Word/exit)>> Hello
Hello->
CMD(Word/exit)>> World!
Not Chained
Hello->
```

<case 2>

```
Hello->
CMD(Word/exit)>> original
Hello->original->
CMD(Word/exit)>> legend
Hello->original->legend->
CMD(Word/exit)>> driver
Hello->original->legend->driver->
CMD(Word/exit)>> rival
Hello->original->legend->driver->rival->
CMD(Word/exit)>>
```

<case 3>

```
Hello->original->legend->driver->rival->
CMD(Word/exit)>> legend
Already Exists
```

2. Implement a program that fills the Binary Search Tree (BST) after passing the Queue. At the beginning of the program, it generates an empty Queue and an empty BST. The conditions of this program are as follows:

- ✓ Only natural number can be input.
- ✓ The maximum size of Queue is 5.
- ✓ If there already exists the same node which has the same value as the input node, in the BST, then the input node will be removed.
- ✓ Deallocate all memory in Queue and BST at the end of the program.

Command

Name	Format	Description
Enqueue	Enqueue <integer>	Make a new node with input factor and add it to Queue.
Dequeue	Dequeue <integer>	Remove nodes from Queue by input factor and insert them to BST. If the factor is great than Queue size, it works like "Dequeue <Queue size>".
Print_Queue	Print_Queue	Print all elements of Queue.
SEARCH	SEARCH <integer>	Find the element in the BST and print.
PRINT	PRINT <method>	Print all elements of BST according to the following traversal method. method: PRE: pre-order IN: in-order POST: post-order
EXIT	EXIT	Program end

ERROR CODE

ERROR CODE	Description
100	Command does not exist
200	Parameters of command are insufficient or flood or wrong
300	Queue is full while running Enqueue
400	Queue is empty while running Dequeue
500	Queue is empty while running Print_Queue
600	Result of SEARCH does not exist
700	BST is empty while running PRINT

Example

```
선택 Microsoft Visual
CMD>> Enqueu 7
Error 100
CMD>> Enqueue a
Error 200
CMD>> Enqueue
Error 200
CMD>> Enqueue 1 3
Error 200
CMD>>
Error 100
```

<case 1>

```
선택 Microsoft Visual
CMD>> Print_Queue
Error 500
CMD>> PRINT IN
Error 700
CMD>> Dequeue 1
Error 400
CMD>> SEARCH 4
Error 600
```

<case 2>

```
선택 Microsoft Visual Studio 디버그 콘솔
CMD>> Enqueue 7
CMD>> Enqueue 4
CMD>> Enqueue 9
CMD>> Print_Queue
7       4       9
CMD>> Dequeue 1
CMD>> PRINT IN
7
CMD>> Enqueue 13
CMD>> Enqueue 11
CMD>> Enqueue 6
CMD>> Enqueue 12
Error 300
CMD>> Print_Queue
4       9       13      11      6
CMD>> Dequeue 3
CMD>> Enqueue 7
CMD>> Dequeue 4
CMD>> Dequeue 1
Error 400
CMD>> Print_Queue
Error 500
CMD>> PRINT PRE
7       4       6       9       13      11
CMD>> PRINT IN
4       6       7       9       11      13
CMD>> PRINT POST
6       4       11      13      9       7
CMD>> SEARCH 4
4is exists
CMD>> EXIT
```

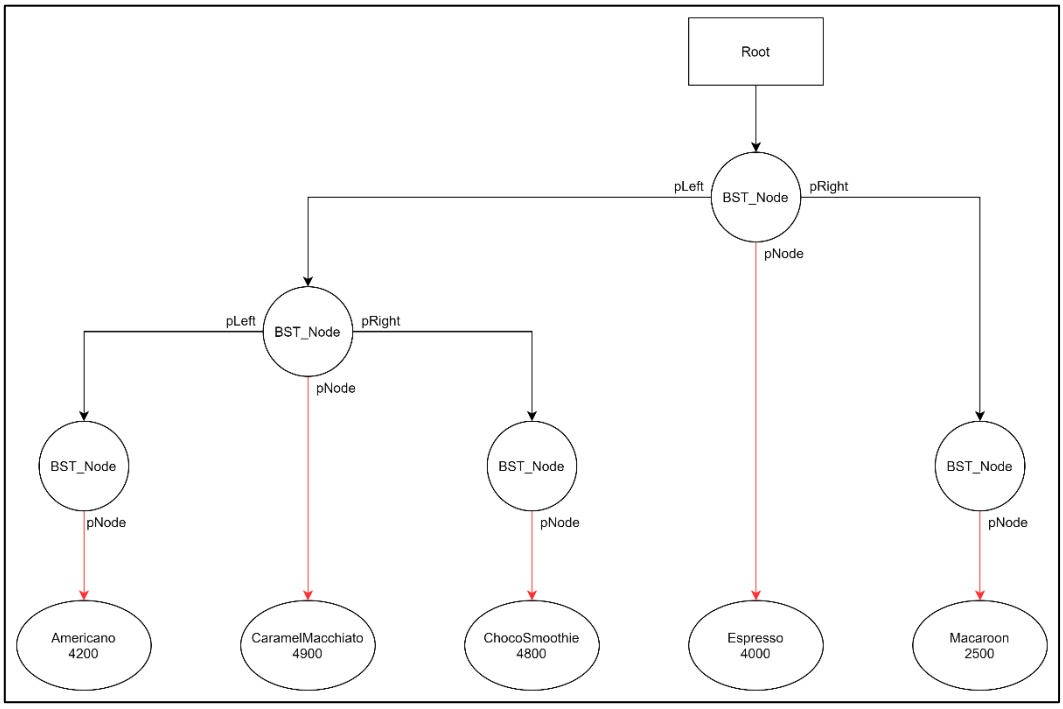
<case 3>

3. Implement a program to manage the café menu using linked list and binary search tree. The program reads from the "menu.txt" that stores café menu information. As shown in the figure below, menu and price are stored in the menu node, and they are connected to linked list in order of price. If prices are same, sort by menu name. Additionally, BST is constructed based on the ascending order of the menu name and bst node points to a menu node. This program can perform the functions of LOAD, PRINT, INSERT, SEARCH, DELETE as shown in the example below.

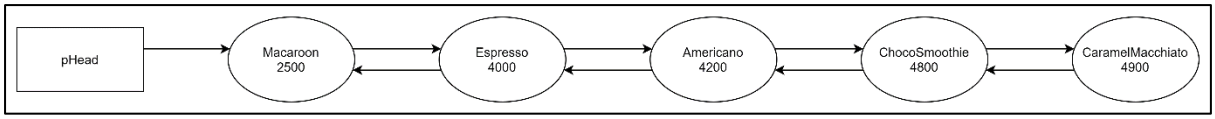
Note:

- ✓ Information in file "menu.txt" is separated by comma.
- ✓ Sort by menu name is case insensitive.
- ✓ If the parameter of the input value in insufficient or flood or the menu already exists, print error.
- ✓ The node itself needs to be changed, not the attribute value in the node.

Description of BST



Description of Linked List



Example of node class

```
class menu_node
{
private:
	char* menu_name;
	int price;
	menu_node* pPrev;
	menu_node* pNext;
public:
	...
};
```

```
class bst_node
{
private:
	menu_node* pNode;
	bst_node* pLeft;
	bst_node* pRight;
public:
	...
};
```

Example

```
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> PRINT
Cafe Menu is Empty
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> SEARCH
Cafe Menu is Empty
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> DELETE
Cafe Menu is Empty
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>>
```

<while menu is empty>

```
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> INSERT
Menu Name: Cappuccino
Price: 4800
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> PRINT
Command list: MENU, PRICE
CMD>> MENU
Print by Name order
Cappuccino      4800

Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>>
```

<INSERT>

```
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> LOAD
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> PRINT
Command list: MENU, PRICE
CMD>> MENU
Print by Name order
Americano               4200
Bagel                   3000
Brownie                 2500
Cappuccino              4800
Caramel Macchiato       4900
Choco Smoothie          4800
ColdBrew                5000
Earlgray Black Tea      3500
Espresso                4000
Grapefruit Ade          5500
Grapefruit Tea          4800
Greentea Smoothie       5400
Horney Bread            5000
Jasmine Tea             4700
Lemon Ade               5500
Lime Tea                5600
Macaroon                2500
Mango Smoothie          5000
Orange Ade              5500
Strawberry Ade          5700
Yogurt Smoothie         4900

Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>>
```

<LOAD, PRINT by MENU>

```
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> PRINT
Command list: MENU, PRICE
CMD>> PRICE
Print by Price order
Brownie                 2500
Macaroon                2500
Bagel                   3000
Earlgray Black Tea      3500
Espresso                4000
Americano               4200
Jasmine Tea             4700
Cappuccino              4800
Choco Smoothie          4800
Grapefruit Tea          4800
Caramel Macchiato       4900
Yogurt Smoothie         4900
ColdBrew                5000
Horney Bread            5000
Mango Smoothie          5000
Greentea Smoothie       5400
Grapefruit Ade          5500
Lemon Ade               5500
Orange Ade              5500
Lime Tea                5600
Strawberry Ade          5700
```

<PRINT by PRICE>

```
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> SEARCH
Menu Name: Americano
Price: 4200
```

```
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> SEARCH
Menu Name: Dutch Coffee
Not in menu
```

<SEARCH>

```
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> DELETE
Menu Name: Espresso
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
```

```
Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>> DELETE
Menu Name: Dutch Coffee
Not in menu
```

<DELETE>

```
Print by Name order
Americano               4200
Bagel                   3000
Brownie                 2500
Cappuccino              4800
Caramel Macchiato       4900
Choco Smoothie          4800
ColdBrew                5000
Earlgray Black Tea      3500
Grapefruit Ade          5500
Grapefruit Tea          4800
Greentea Smoothie       5400
Horney Bread            5000
Jasmine Tea             4700
Lemon Ade               5500
Lime Tea                5600
Macaroon                2500
Mango Smoothie          5000
Orange Ade              5500
Strawberry Ade          5700
Yogurt Smoothie         4900

Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>>
```

```
Print by Price order
Brownie                 2500
Macaroon                2500
Bagel                   3000
Earlgray Black Tea      3500
Americano               4200
Jasmine Tea             4700
Cappuccino              4800
Choco Smoothie          4800
Grapefruit Tea          4800
Caramel Macchiato       4900
Yogurt Smoothie         4900
ColdBrew                5000
Horney Bread            5000
Mango Smoothie          5000
Greentea Smoothie       5400
Grapefruit Ade          5500
Lemon Ade               5500
Orange Ade              5500
Lime Tea                5600
Strawberry Ade          5700

Command list: LOAD, PRINT, INSERT, SEARCH, DELETE, EXIT
CMD>>
```

<PRINT after DELETE>

4. Implement a Blackjack card game using linked lists. At the beginning of the program, it generates 52 nodes that are the set of playing card. There are 4 linked list named "Deck", "Discard_tray", "Dealer", and "Player" for manage nodes. The "Deck" is connected to the generated nodes, and the others are empty. After that, the program can receive commands "game", "shuffle", and "exit".

If received "shuffle", place all cards of "Discard_tray" on top of "Deck", and select random cards (1-51) from bottom of "Deck". Place a random number (1 to number of selected cards) of cards on the "Deck" from the top of the selected cards and repeat until the number of selected cards is zero. The "shuffle" command takes a natural number as argument and repeats the number of arguments received.

If received "game", start the game. The game can only be started when all 52 cards are connected to "Deck". The rules of the game are as follows:

1. "bet" / "end" command to game match start or game end.
2. The player and dealer are dealt 2 cards each on the "Deck".
3. Reveal the player's 2 cards and the dealer's 1 card.
4. The player can take one more card by "hit" or stop by "stand".
5. Lose immediately, if the sum of the values on the card exceeds 21 by "hit".
    - Ace cards can convert to 1 or 11.
    - Face cards (Jack, Queen, and King) are all convert to 10.
6. When the player stops, the dealer reveals the card, "hit" for 16 or less, "stand" for 17 or higher, and repeats until "stand".
    - Dealer takes "hit" or "stand" automatically.
    - Rule 5 also applies to the dealer.
7. The winner is the one with the sum of the card values close to 21 without exceeding 21, If same then draw.
8. The cards of the player and dealer are moved to "Discard_tray".
9. Repeat <1-8> while the number of cards remaining in "Deck" is 13 or more.

If received "exit", program end.

Ref. https://ko.wikipedia.org/wiki/%EB%B8%94%EB%9E%99%EC%9E%AD

Example

```
Command list(game/shuffle/exit)
CMD>>
```

<program start>

```
Command list(game/shuffle/exit)
CMD>> gam
Wrong Command!
```

<insert wrong command>

```
Command list(game/shuffle/exit)
CMD>> shuffle 192
```

<shuffle command>

```
Command list(game/shuffle/exit)
CMD>> game
```

<game command>

```
Command list(bet/end)
CMD>>
```

```
Command list(bet/end)
CMD>> bet
```

<game base>

```
Dealer Cards :  ?       S8
Player Cards :  C5      D6
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  ?       S8
Player Cards :  C5      D6      DA
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  D4      S8
Player Cards :  C5      D6      DA      DJ
Player Lose!
Command list(bet/end)
CMD>>
```

<lose scenario 1 (exceed 21)>

```
Dealer Cards :  ?       CK
Player Cards :  S5      H5
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  ?       CK
Player Cards :  S5      H5      S3
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  ?       CK
Player Cards :  S5      H5      S3      D4
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  H3      CK
Player Cards :  S5      H5      S3      D4
```

```
Dealer Cards :  H3      CK      C8
Player Cards :  S5      H5      S3      D4
Player Lose!
Command list(bet/end)
CMD>>
```

<lose scenario 2-1 (less than dealer)>

```
Dealer Cards :  ?       CA
Player Cards :  HK      S2
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  ?       CA
Player Cards :  HK      S2      H6
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  HQ      CA
Player Cards :  HK      S2      H6
```

```
Dealer Cards :  HQ      CA
Player Cards :  HK      S2      H6
Player Lose!
Command list(bet/end)
CMD>>
```

<lose scenario 2-2 (less than dealer)>

```
Dealer Cards :  ?       H9
Player Cards :  CK      H5
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  ?       H9
Player Cards :  CK      H5      H4
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  HQ      H9
Player Cards :  CK      H5      H4
```

```
Dealer Cards :  HQ      H9
Player Cards :  CK      H5      H4
 Draw...
Command list(bet/end)
CMD>>
```

<draw scenario 1>

```
Dealer Cards :  ?       D2
Player Cards :  S9      CQ
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  H7      D2
Player Cards :  S9      CQ
```

```
Dealer Cards :  H7      D2      DK
Player Cards :  S9      CQ
 Draw...
Command list(bet/end)
CMD>>
```

<draw scenario 2>

```
Dealer Cards :  ?       C7
Player Cards :  D5      D2
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  ?       C7
Player Cards :  D5      D2      S7
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  ?       C7
Player Cards :  D5      D2      S7      HA
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  H2      C7
Player Cards :  D5      D2      S7      HA
```

```
Dealer Cards :  H2      C7      C5
Player Cards :  D5      D2      S7      HA
```

```
Dealer Cards :  H2      C7      C5      C9
Player Cards :  D5      D2      S7      HA
Player Win!
Command list(bet/end)
CMD>>
```

<win scenario 1 (exceed 21)>

```
Dealer Cards :  ?       D7
Player Cards :  D6      C4
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  ?       D7
Player Cards :  D6      C4      S5
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  ?       D7
Player Cards :  D6      C4      S5      H4
Command list(hit/stand)
CMD>>
```

```
Dealer Cards :  HQ      D7
Player Cards :  D6      C4      S5      H4
```

```
Dealer Cards :  HQ      D7
Player Cards :  D6      C4      S5      H4
Player Win!
Command list(bet/end)
CMD>>
```

<win scenario 2-1 (greater than dealer)>

```
Dealer Cards :  S5      C4      SJ
Player Cards :  CA      C7      S3
Player Win!
Command list(bet/end)
CMD>>
```

<win scenario 2-2 (greater than dealer)>

```
Dealer Cards :  SK      HJ
Player Cards :  D8      H4      D3      H5
 Draw...
Not Enough Cards
Command list(game/shuffle/exit)
CMD>>
```

<number of cards remaining in "Deck" is less than 13>

```
CMD>> game
Not Enough Cards
Command list(game/shuffle/exit)
CMD>>
```

<game only can start with 52 cards>

5. Write the Russian Roulette program using linked list. The flow of program is as follows:

1. Create a Revolver, which is circular linked list.
2. Insert one bullet to Revolver and "rotate".
3. "shoot" / "rotate" command to shoot or rotate.
4. If receive "shoot" command then check the bullet, if exists, then print "You Died..." and program end.
5. If not exists, then print "You Survived!" and repeat 3.

Conditions:

- ✓ Revolver is connected to 6 nodes, which are created in Revolver's constructor.
- ✓ The "shoot" command checks Revolver's head and moves the head to the next node.
- ✓ "rotate" means that Revolver's head moves to the next node by a random number.
- ✓ After "rotate" clean the screen using 'system("cls")'.

Example

```
Command list(shoot/rotate)
CMD>>
```

<base>

```
Command list(shoot/rotate)
CMD>> shoot
You Survived!
CMD>>
```

<shoot command>

```
Command list(shoot/rotate)
CMD>> reload
Wrong Command!
CMD>>
```

<wrong command>

```
Command list(shoot/rotate)
CMD>> shoot
You Survived!
CMD>> shoot
You Survived!
CMD>> shoot
You Survived!
CMD>> shoot
You Survived!
CMD>> shoot
You Died...
```

<case 1>

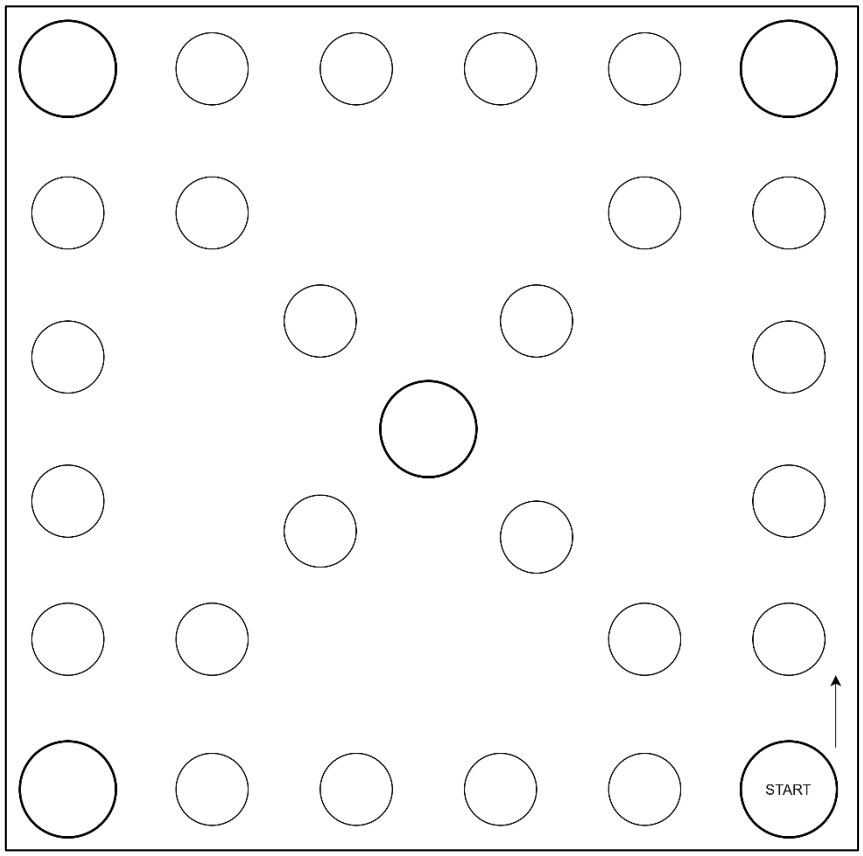
```
Command list(shoot/rotate)
CMD>> shoot
You Survived!
CMD>> shoot
You Died...
```

<case2>

```
Command list(shoot/rotate)
CMD>> shoot
You Survived!
CMD>> shoot
You Survived!
CMD>> shoot
You Survived!
CMD>> rotate
```

<case3>

6. Implement a Yut Nori board game using linked lists. In this game, 2 players each has 4 horses, and the player who finishes all the horses first wins. The board is composed as shown figure 1, and the horse from starting point returns to the starting point, and the race is completed. The horse moves according to the result of throwing 4 yuts, and the result is the same as table 1.



<figure 1 board>

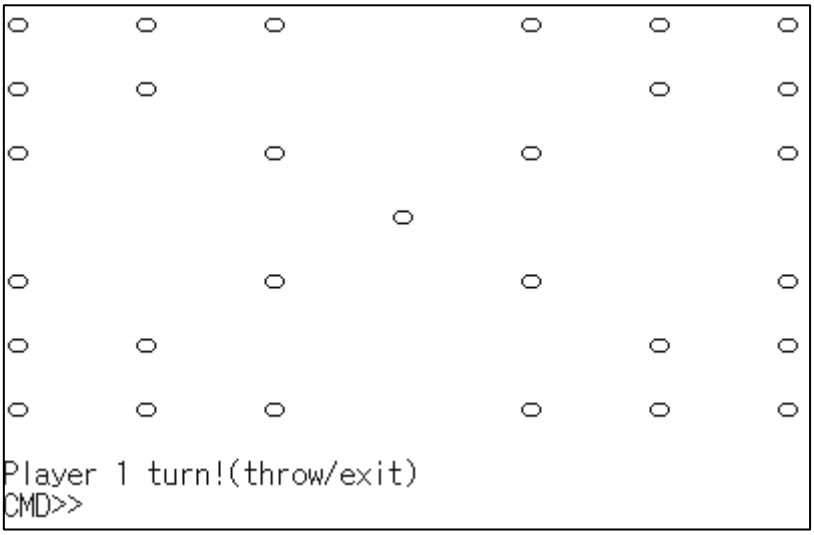
State	Action
One yut is front side Three yuts are back side	One horse goes forward one step
Two yuts are front side Two yuts are back side	One horse goes forward two steps
Three yuts are front side One yut is back side	One horse goes forward three steps
Four yuts are front side	One horse goes forward four steps Throwing the yuts once again
Four yuts are back side	One horse goes forward five steps Throwing the yuts once again
One marked yut is front side Three yuts are back side	One horse goes one step back If there is no horse on the board, nothing happens

<table1 >

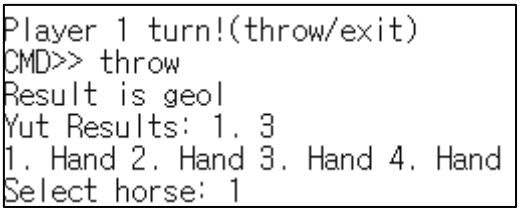
The horse moves along the outer path but horses that begin their movement on a large, thickly marked circle moves to a short path. The yuts consist of 3 unmarked yuts and 1 marked yut. A yut has a front and a back side, and the probability of getting a front side by throwing yut is 60%. During the game, if the horses of the same player overlap, player can carry the horses and move them together. If it overlaps with another player's horse, the caught horse is returned to the caught player, and the player who caught the opponent's horse can throw the yut again(But if caught the opponent's horse with 'yut', then cannot throw yuts again.).

Ref. https://ko.wikipedia.org/wiki/%EC%9C%B7%EB%86%80%EC%9D%B4

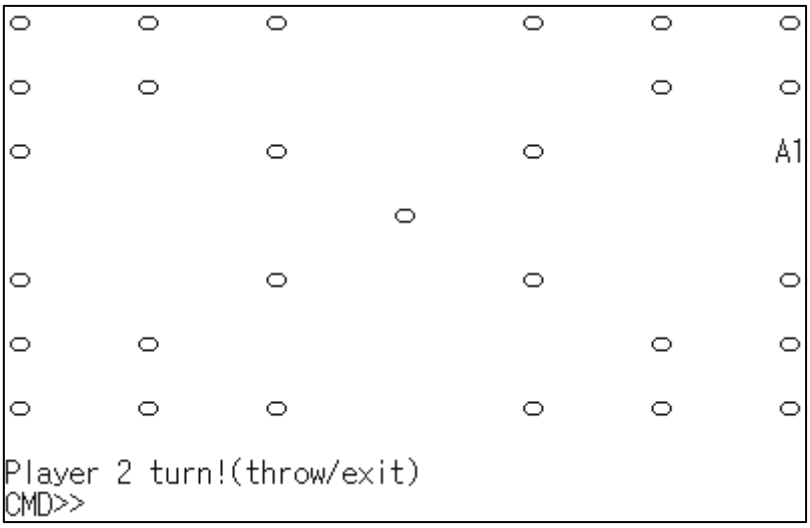
Example



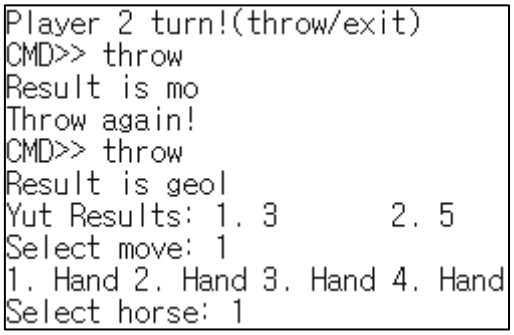
(1) Program base (Start point is the bottom right corner and go to up)

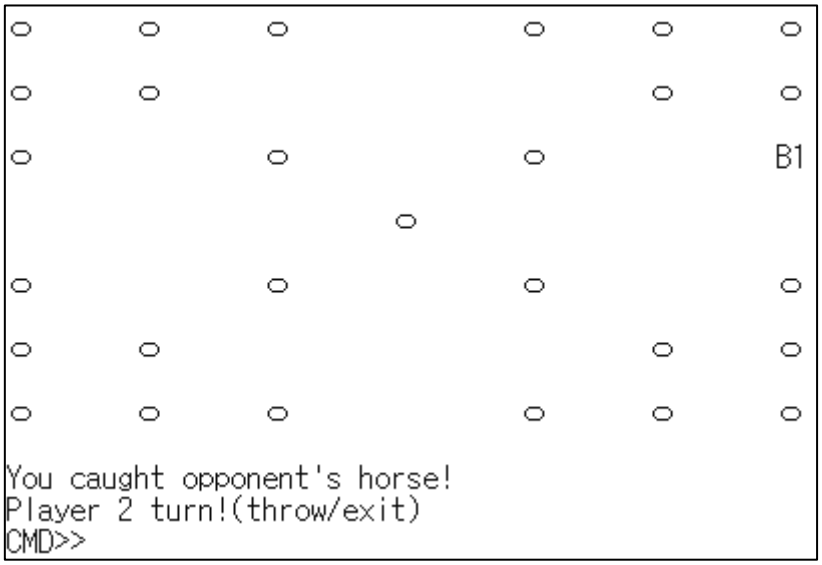


(2) Throwing the yuts and move horse.



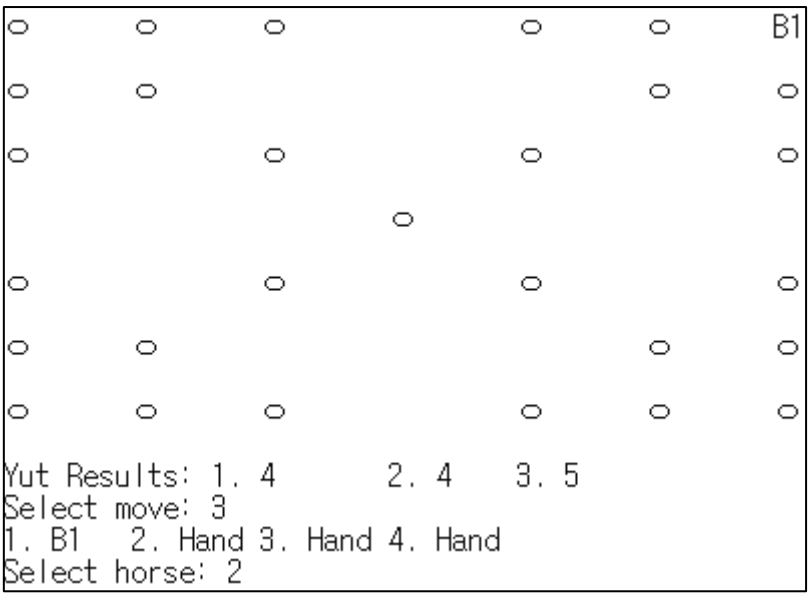
(3) Result of (2)





(4) Result of the yuts is 'yut' or 'mo', or caught the opponent's horse then throw yuts again.

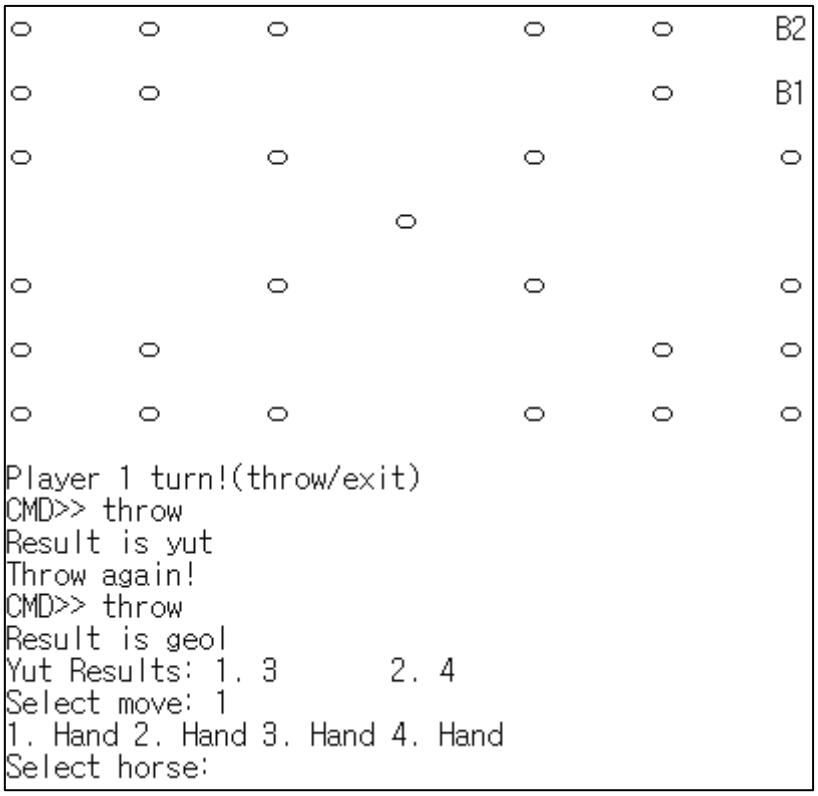
```
You caught opponent's horse!
Player 2 turn!(throw/exit)
CMD>> throw
Result is yut
Throw again!
CMD>> throw
Result is yut
Throw again!
CMD>> throw
Result is gae
Yut Results: 1. 2       2. 4    3. 4    4. 5
Select move: 1
1. B1   2. Hand 3. Hand 4. Hand
Select horse: 1
```



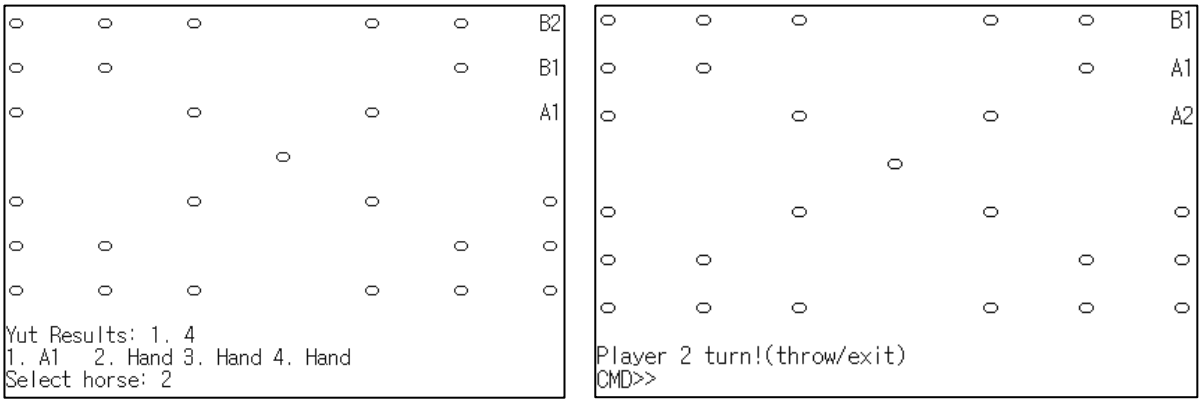
(5) Select the result to move, move the horse, and raise when other horses overlap.

```
Yut Results: 1. 4       2. 4
Select move: 1
1. B1   2. Hand 3. Hand
Select horse:
```

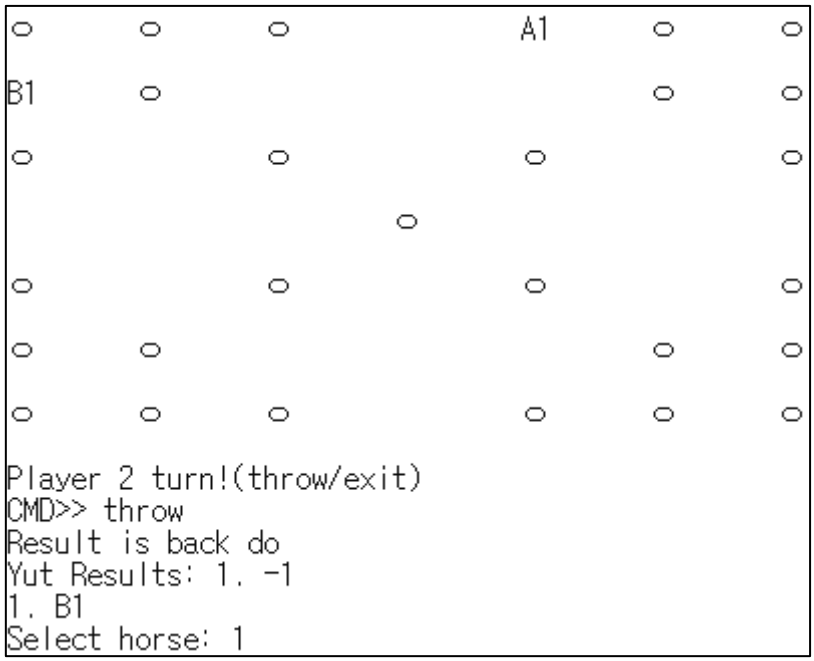
(6) If hold up your horse, it moves together.

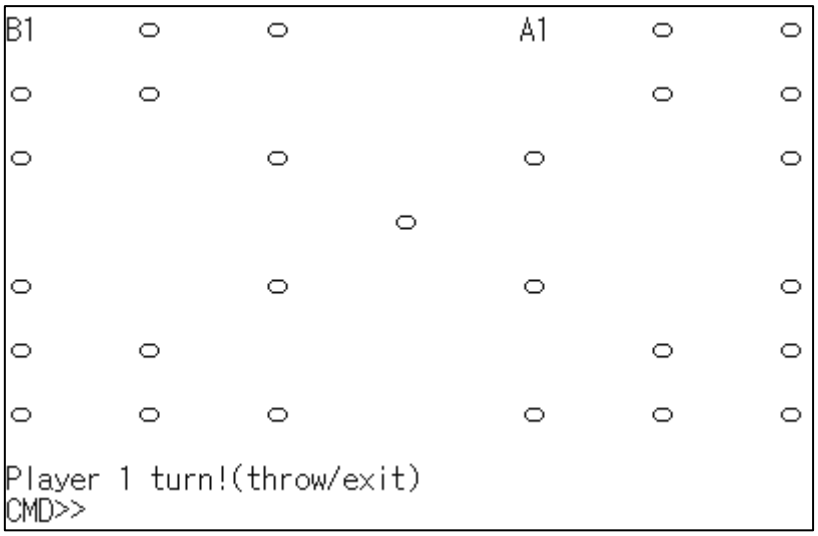


(7) The caught horses are returned to the player's hand.

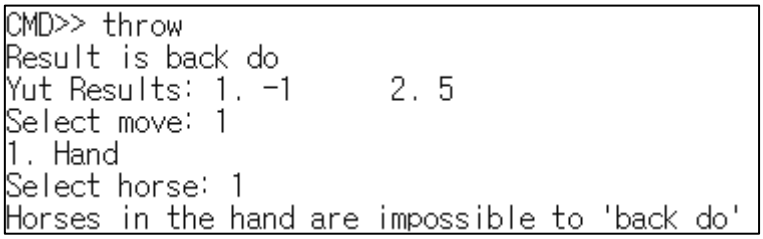


(8) If caught the opponent's horse with 'yut', then cannot throw yuts again.

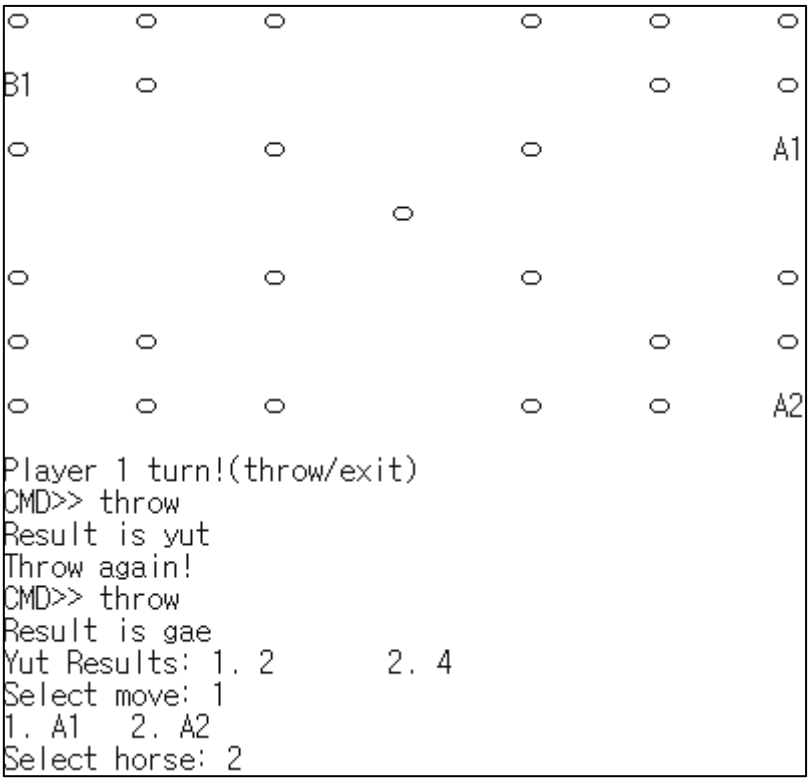


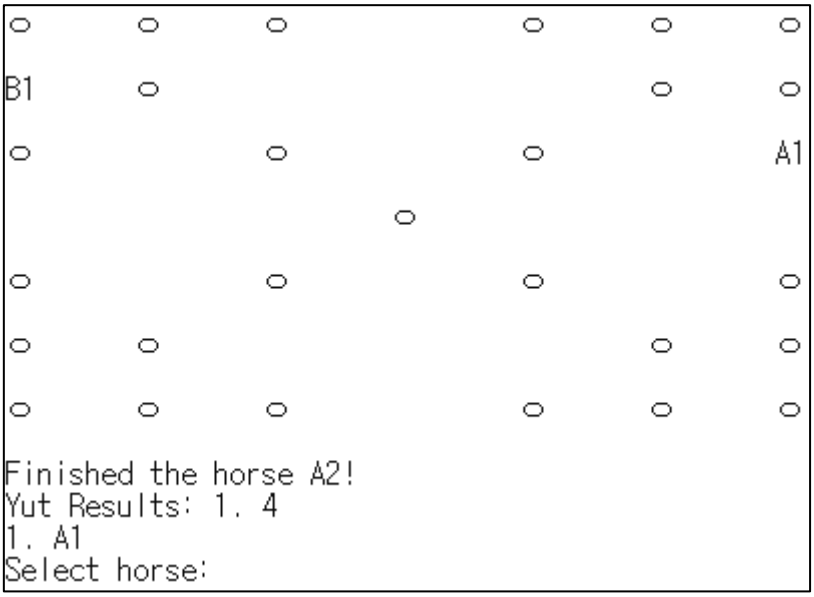


(9) "back do" moves back one step.

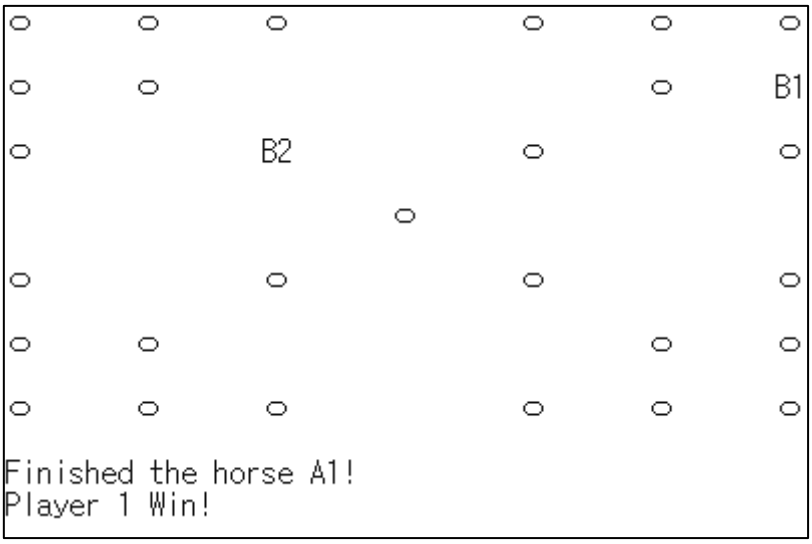


(10) Horses in a player's hand cannot be moved "back do".





(11) When the horse that has returned to the starting point advances, it will finish.



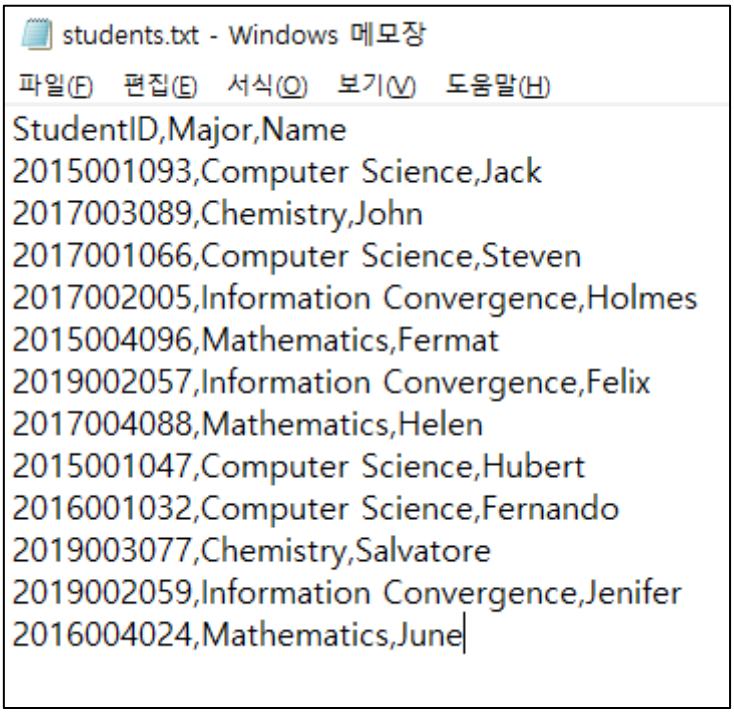
(12) The player who finishes all horses first wins.

7. Write a program that manages student's information using linked lists and Template. The program reads student information from a file "students.txt". The student information is composed of "ID", "Major", and "Name", and is divided into comma. 'Student_Node' has student information and 4 additional pointers. Using these pointers create three 2d linked lists and one 1d linked list. One 1d linked list is sorted by input order. One of the three 2d linked lists is sorted by entrance year in ascending using ID, other is sorted by Major name in ascending using Major, and the other is sorted by Alphabet in ascending using Name. For 'ID' case, all the nodes in the same year should be sorted in ascending order of ID. In addition, the nodes in the same Major Name for Major list should be sorted in ascending order of ID. Implement by referring to the following:

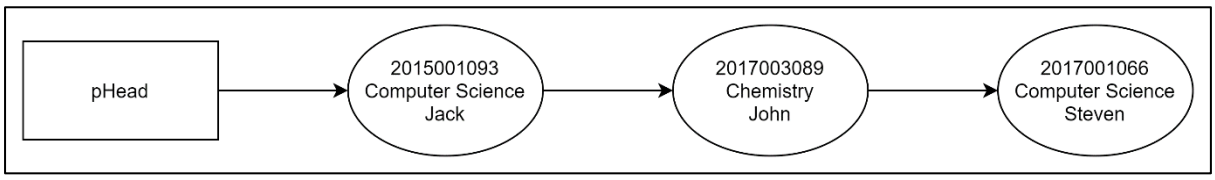
* Do not make duplicate node, which have same information.

* You must use Template to implement 2d linked lists.

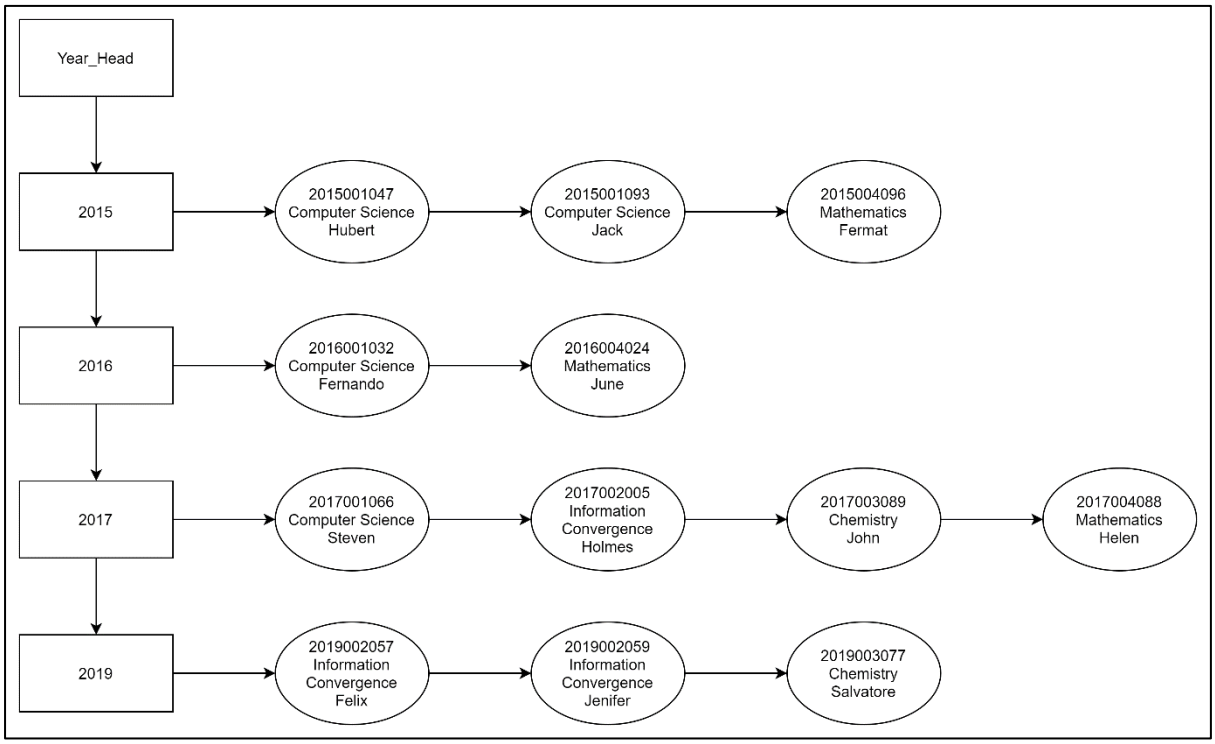
* Assume there are no invalid inputs.



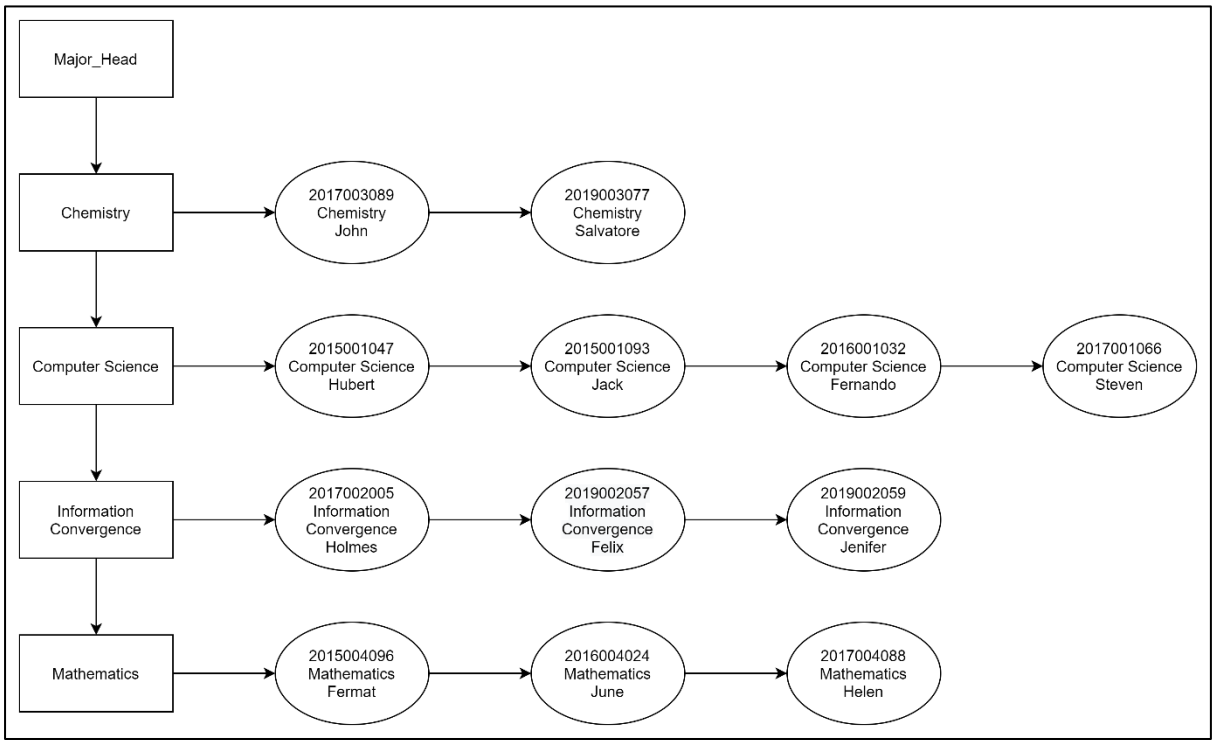
<example of 'students.txt'>



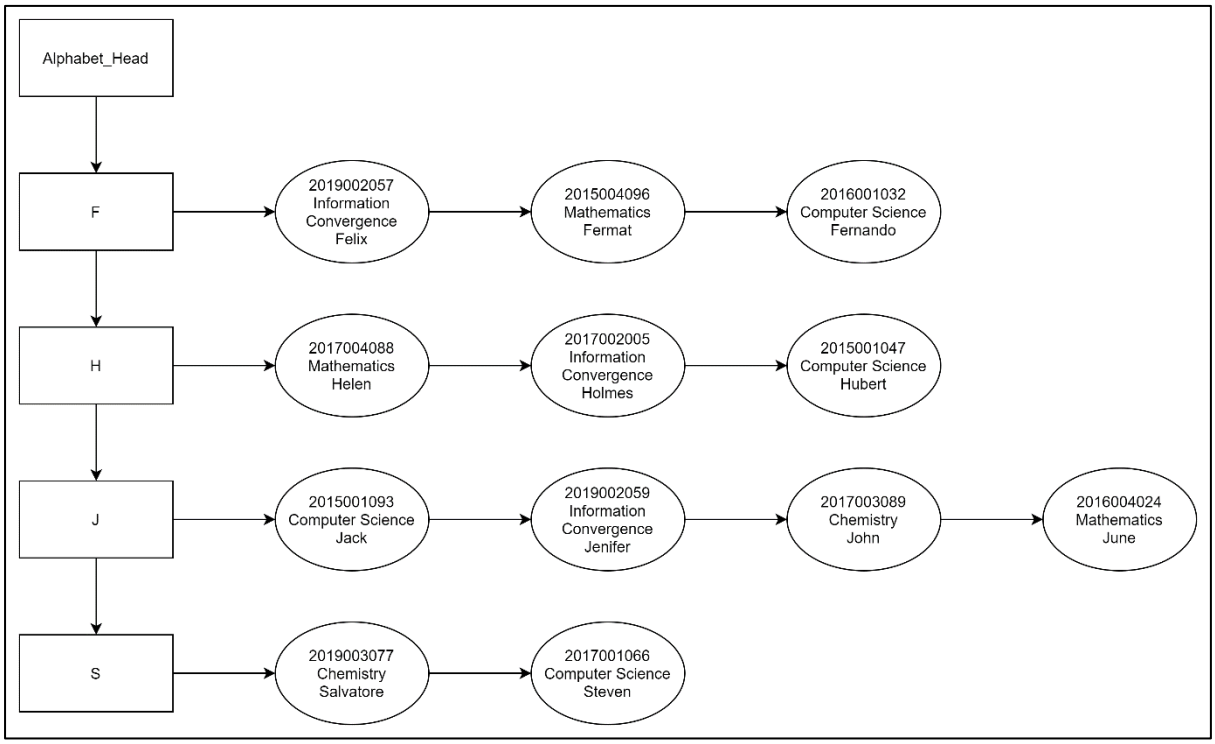
<description of input order>



<description of ID>



<description of Major>



<description of Name>

```
1. Print File
2. Print by ID
3. Print by Major
4. Print by Name
5. Program End
CMD>>
```

<base>

```
Microsoft Visual Studio 디버그 콘솔
Print File
============================================================
StudentID       Major                                   Name
2015001093      Computer Science                        Jack
2017003089      Chemistry                               John
2017001066      Computer Science                        Steven
2017002005      Information Convergence                 Holmes
2015004096      Mathematics                             Fermat
2019002057      Information Convergence                 Felix
2017004088      Mathematics                             Helen
2015001047      Computer Science                        Hubert
2016001032      Computer Science                        Fernando
2019003077      Chemistry                               Salvatore
2019002059      Information Convergence                 Jenifer
2016004024      Mathematics                             June
============================================================
```

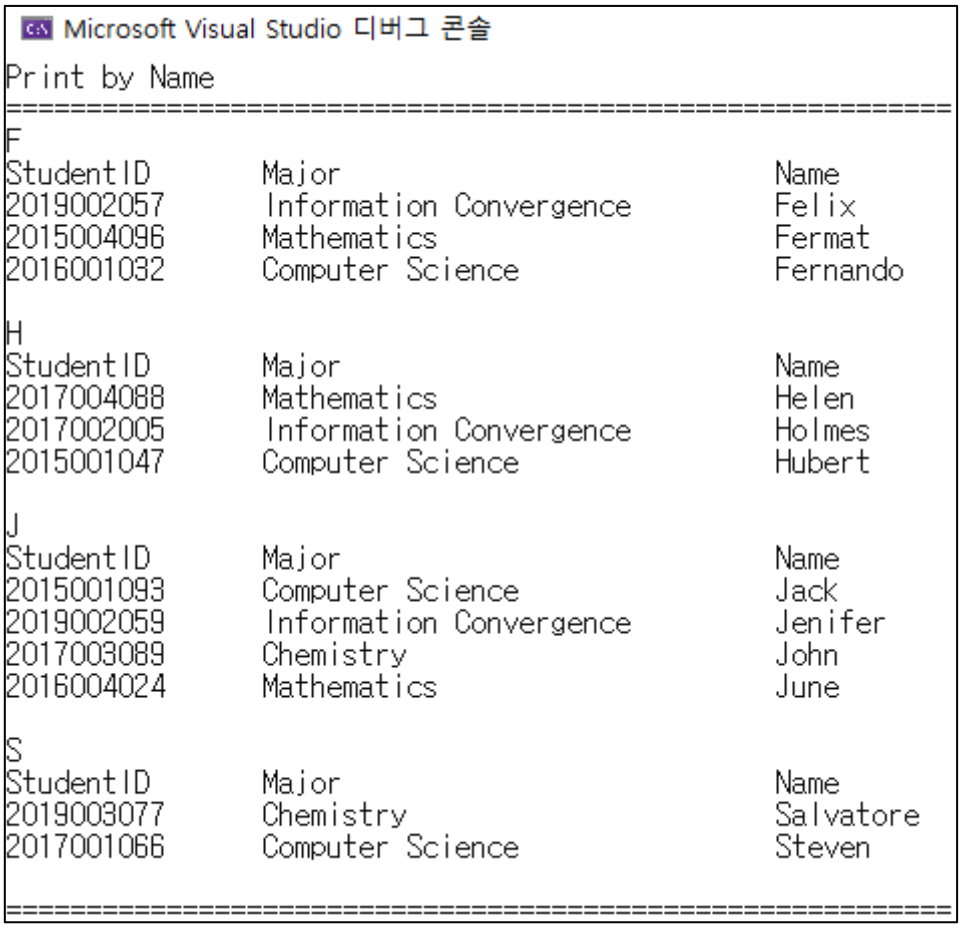
<print by input order>

```
Microsoft Visual Studio 디버그 콘솔
Print by ID
============================================================
2015
StudentID       Major                           Name
2015001047      Computer Science                Hubert
2015001093      Computer Science                Jack
2015004096      Mathematics                     Fermat

2016
StudentID       Major                           Name
2016001032      Computer Science                Fernando
2016004024      Mathematics                     June

2017
StudentID       Major                           Name
2017001066      Computer Science                Steven
2017002005      Information Convergence         Holmes
2017003089      Chemistry                       John
2017004088      Mathematics                     Helen

2019
StudentID       Major                           Name
2019002057      Information Convergence         Felix
2019002059      Information Convergence         Jenifer
2019003077      Chemistry                       Salvatore

============================================================
```

<print by ID>

```
Microsoft Visual Studio 디버그 콘솔
Print by Major
============================================================
Chemistry
StudentID       Major                           Name
2017003089      Chemistry                       John
2019003077      Chemistry                       Salvatore

Computer Science
StudentID       Major                           Name
2015001047      Computer Science                Hubert
2015001093      Computer Science                Jack
2016001032      Computer Science                Fernando
2017001066      Computer Science                Steven

Information Convergence
StudentID       Major                           Name
2017002005      Information Convergence         Holmes
2019002057      Information Convergence         Felix
2019002059      Information Convergence         Jenifer

Mathematics
StudentID       Major                           Name
2015004096      Mathematics                     Fermat
2016004024      Mathematics                     June
2017004088      Mathematics                     Helen

============================================================
```

<print by Major>



<print by Name>

Due date: 2021/05/28,23:59:59

Delay date: 2021/05/29,11:59:59

If you have questions about Assignment 3, contact below.

micky1996@naver.com (이종학, 새빛관 909호)