

<1> [Class-Inheritance] The village in Nuri Island, which Dragon-sun protects, is now in danger because Dragon-sun got cursed by the Great Wizard. But the brave villagers have a will to fight and 5 villagers got ready for the battle.

There are four classes, Peasant Army, Sword Master, Archer, and Warlock in the village. Each class has different attack patterns and abilities like the following diagram and tables.

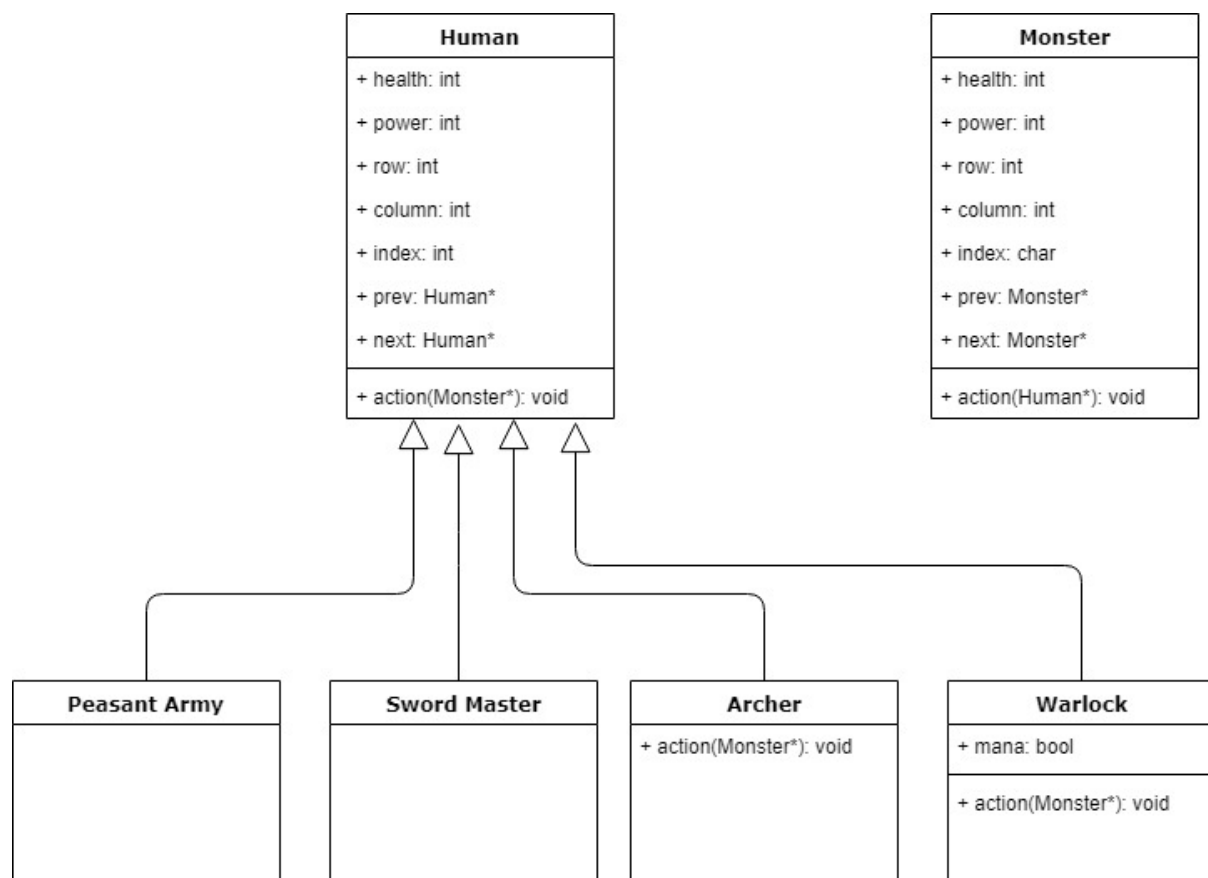


Figure 1 Class Diagram

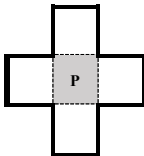
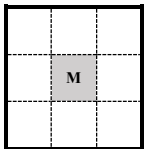
Human	Monster
Health	Health: 20
Power	Power: 7
<b>Coordinates:</b> Row Column	<b>Coordinates:</b> Row Column
Index	Index
Human* prev	Monster* prev
Human* next	Monster* next
Action(Monster*)  If there are any opponent instances in the attack range, attack that instance. If not, move toward the closest opponent instance  This function will reduce the health of the target instance as much as its own power  The shape of the attack range is like below  	Action(Human*)  If there are any opponent instances in the attack range, attack that instance. If not, move toward the closest opponent instance  This function will reduce the health of the target instance as much as its own power  The shape of the attack range is like below  

Figure 2 Class table

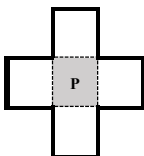
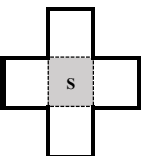
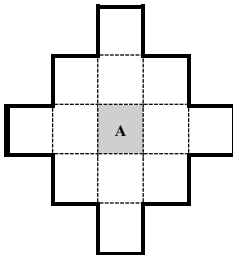
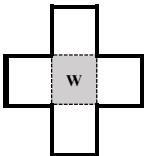
Peasant Army	Sword Master	Archer	Warlock
Health: 5	Health: 10	Health: 7	Health: 3
Power: 5	Power: 10	Power: 10	Power: 10
			<b>Mana:</b> True This value will be inverted after each turn It means Warlock can attack once for two turns
Action(Monster*)  The shape of the attack range is like below  	Action(Monster*)  The shape of the attack range is like below  	Action(Monster*)  <b>Ranged attack</b> The shape of the attack range is like below Can attack any monster in the attack range  	Action(Monster*)  <b>Wide-area attack</b> The shape of the attack range is like below Can attack up to 4 monsters at the same time  

Figure 3 Class table of Human

Assignment #3, Object Oriented Programming. 2019 Spring Semester

Due date: 2019/05/31 23:59

Implement the program that simulates and shows the results of the battle between 5 villagers and 5 monsters until there are no more instances in any camps. Five villagers consist of one Peasant Army, two Sword Master, one Archer, and one Warlock.

The result screen must be like below format

```
##### Result of round 1 #####
<Human>
```

```
1
2
3
4
5
```

```
<Monster>
```

```
A
B
C
D
E
```

```
##### Result of round 2 #####
```

```
<Human>
```

```
2
4
```

```
<Monster>
```

```
A
B
C
D
E
```

```
Human 1 has died
```

```
Human 3 has died
```

```
Human 5 has died
```

```
##### Result of round 3 #####
```

```
<Human>
```

```
4
```

```
<Monster>
```

```
A
B
C
E
```

```
Human 2 has died
```

```
Monster D has died
```

```
##### Result of round 4 #####
```

```
<Human>
```

```
4
```

```
<Monster>
```

```
A
B
C
E
```

```
##### Result of round 5 #####
```

```
<Human>
```

```
<Monster>
```

```
A
B
C
```

```
Human 4 has died
```

```
Monster E has died
```

```
##### Final result of battle #####
```

```
Human has been defeated
```

Your program should follow the below constraints.

- Follow the class inheritance structure as the above class diagram (Figure 1)
  - Abilities like health, power, and coordinates (row, column) will be inherited
  - Action function should be overridden for different action patterns
- Each camp (human and monster) has 5 instances.
  - Instances for each camp are handled with double linked lists (Each camp has their own list)

Index	Instance
1	Peasant Army
2	Sword Master
3	Sword Master
4	Archer
5	Warlock

Index	Instance
A	Monster
B	Monster
C	Monster
D	Monster
E	Monster

**Figure 4 List for Each camp**

- Do not use libraries related to double linked list
- The dead instance should be deleted from the list (An instance whose health is lower than 0 will die)
- The battlefield is 5 x 5 array
  - Each instance has its coordinates for the battlefield

cell (0,0)	(0,1)	(0,2)	(0,3)	(0,4)	Row0
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	Row1
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	Row2
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	Row3
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	Row4
Column0	Column1	Column2	Column3	Column4	

**Figure 5 Coordinates**

- Multiple instances can have the same coordinates
- Instances are placed like the below figure at the beginning

Human 1				Monster A
Human 2				Monster B
Human 3				Monster C
Human 4				Monster D
Human 5				Monster E

**Figure 6 Initial State**

- Each round consists of two turns
  - Each camp is given a turn to attack or move
  - All instances of each camp can attack (or move) in their turn
  - After the end of one camp's turn, the other camp takes a turn
  - Human takes a turn first
  - For example, on Human camp's turn, every human instance can attack or move
- Action Rule for each instance
  - If there are any opponents in the attack range, instance attacks that opponents (attack priority is distance order)
  - If not, move toward the closest opponent cell by cell (move horizontally first)
  - If there are more than 1 instances to attack or move toward, choose the instance which has a lower index
- Battle will end when there are no instances in any camps
  - Each instance has its own index for displaying the result of the battle

<2> (Class, Linked list) Implement a simple cache system which stores words read from a file 'sentences.txt' using LRU(Least recent used) algorithm. Cache has a structure of single linked list composed of maximum five 'CacheNode' classes.


 Sentences.txt - 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)  
Apple is apple pie. Grape juice and orange

Fig. Example of the input file

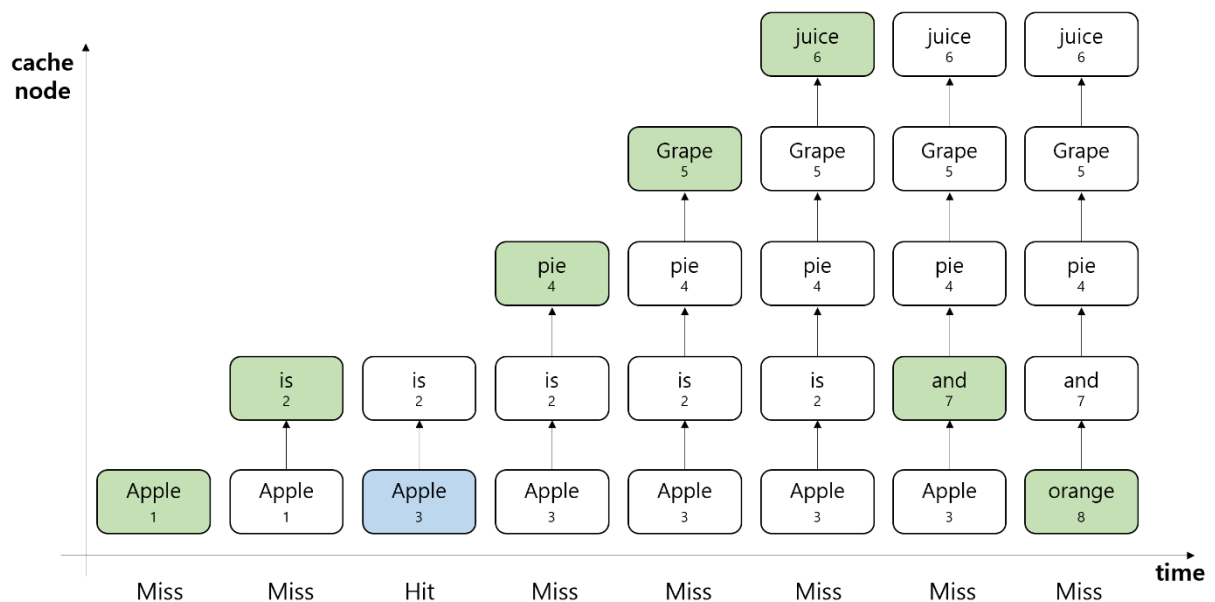


Fig. Example of caching the words from the input file

A description of CacheNode and CacheManager class is given below. CacheNode class and CacheManager class must be used and all the given variables must be used also.

**[class CacheNode]**

- The node of cache linked list has CacheNode type.
- Member variables (Another data type can be used)
  - Char\*/String Word
    - Stores the word
  - int Timestamp
    - Stores the timestamp value of the word
  - CacheNode\* NextNode
    - Stores address of next node of the cache linked list

**[class CacheManager]**

- CacheManager class manages cache system.
- Check whether the word is in the cache, and performs an operation according to the determination result(hit or miss).
- Member variables
  - CacheNode\* HeadNode
    - Address of the head node of the linked list

Implement a program that accesses the words stored in the input file in order and caches each word according to the following method.

#### [Caching rule]

At the beginning of program execution, the cache (linked list) is empty.

When caching, words are not case sensitive.

A word in an input file consists of one or more consecutive alphabets (upper and lower case).

Words in an input file can be separated by one or more consecutive non-alphabet characters (space character, new line character, period).

If the same word as the word from input file exists in linked list,

- Print message as 'Hit'
- Update variable 'timestamp' of that node to the current timestamp value.
  - Timestamp : The sequence number of words when it occurs in the file

Else,

- Print message as 'Miss'
- Create an object of 'CacheNode'
- Save the current word read from input file in a variable 'Word' of the new node
- Save the current timestamp value in a variable 'Timestamp' of the new node.
- If cache is not full,
  - Add the new node to the end of the linked list.
- If cache is full,
  - Replace the node that stores the word that occurred the longest ago in the linked list with a new node. (LRU algorithm)

After every caching, print out all the words with timestamp in the linked list.

#### [Output Example]

```
Miss Apple:1
Miss Apple:1 is:2
Hit Apple:3 is:2
Miss Apple:3 is:2 pie:4
Miss Apple:3 is:2 pie:4 Grape:5
Miss Apple:3 is:2 pie:4 Grape:5 juice:6
Miss Apple:3 and:7 pie:4 Grape:5 juice:6
Miss orange:8 and:7 pie:4 Grape:5 juice:6
```



<3> [Class, Linked list] There are 100 billion neurons in the brain. Among many neurons responsible for arithmetic computation, the connection of particular nine neurons can be disconnected according to the certain rules. Implement a program that simulates the operation of nine neurons according to the rules below.

### [Structure of Neurons]

- The structure of the nine neurons is a linked list
- Each neuron that constitutes a linked list is a class 'Neuron'.

[class Neuron]

- Member variables (You can use another data type)
  - int Value : integer value N,  $0 \leq N \leq 9$
  - Char\*/String Op : One of four operators of addition(add), subtraction(sub), multiplication(mul), and division(div)
  - Neuron\* RightNeuron : address of forward Neuron object connected with current neuron among the neurons of the next layer.
  - Neuron\* DownNeuron : address of forward Neuron object connected with current neuron among the neurons of the same layer.
- Initial state of the nine neurons are shown in the figure below.

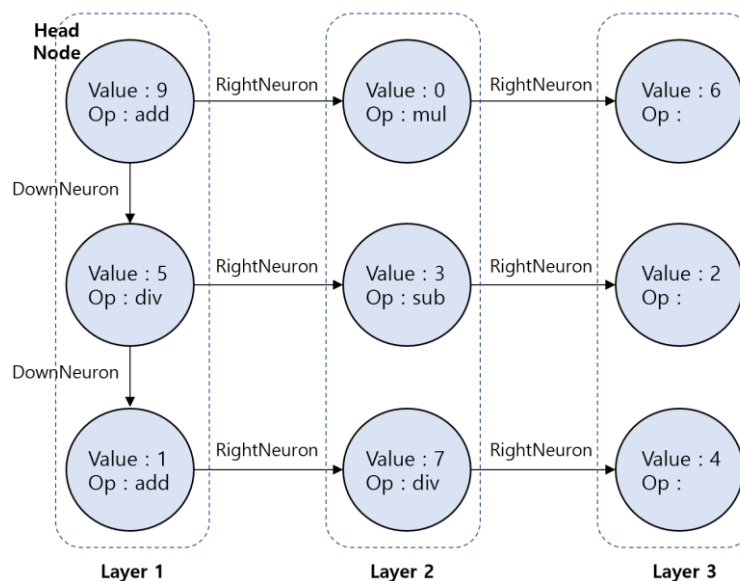


Fig. Initial state of neurons

### [Operating rules of Neurons]

#### 1. Initial state (First state)

- Create nine neurons, and construct a list as shown in the figure above.
- Member variables 'Value' and 'Op' is set as figure above.

#### 2. State update

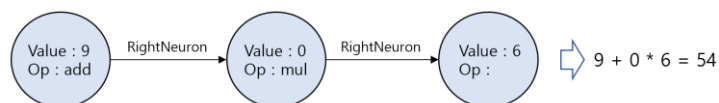
- When the state is updated, each inter-layer connection is broken with a 10% probability.
- Total three state updates are performed in order.
  - First state -> Second state, Second state -> Third state, Third state -> Fourth state
- The connection break is determined as shown in the following code.

```
srand(1);  
break = rand()%10;    //connection will be broken if value of break is zero.
```

- Seed of rand operation should be given with srand(1) for deciding connection will be broken or not.

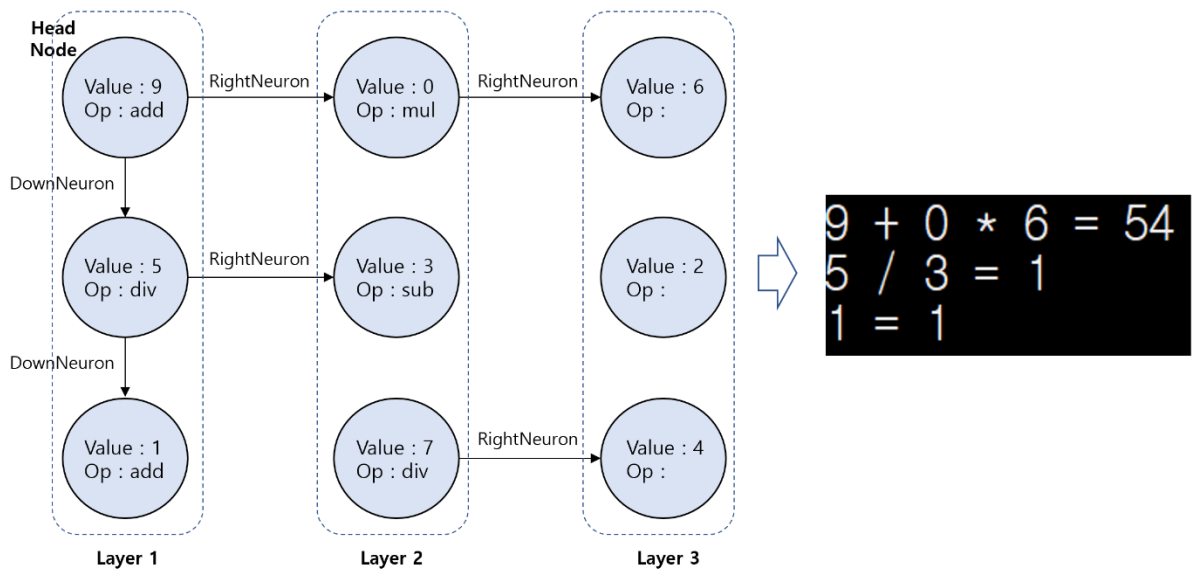
#### 3. Final state (Fourth state)

- In the fourth state, calculation will be executed horizontally from each neuron of first layer until there is no connection
- The horizontal calculation between the layers is proceeds sequentially from the previous layer.



- Print the expression and answer.
- Do not consider the priority of operators.
- Discard the remaining of division.
- If there is no connection between first layer and second layer, display the number of first neuron only.

[Output Example]



Due date: 2019/05/31 23:59

<4> [MFC] A little Wonny wants to do something special because of the boring daily routines. Someday, Wonny read about "Eratosthenes' Sieve" and got some inspiration from that.

So, Wonny placed an array of cards and flipped them according to special rules. In the end, Wonny made "Wonny's Sieve"

"I will flip the cards multiple of 2 at first

and I will flip the cards multiple of 3,

and I will keep doing this to the last cards"

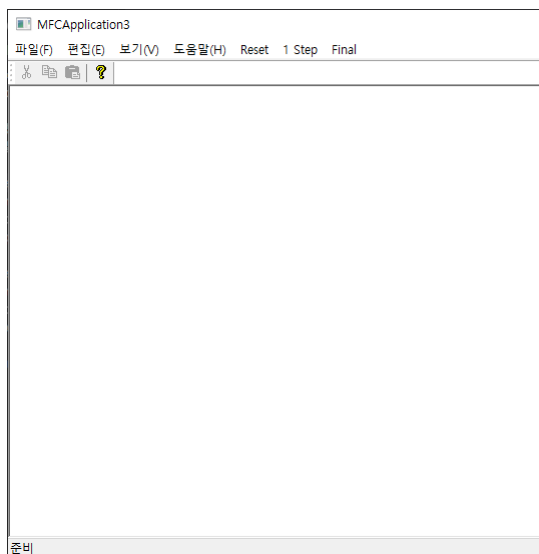
Implement the program that shows the progress of making Wonny's Sieve. Your program should follow the below constraints.

- Implemented with MFC
- Display cards using 52 front images and one back image
- Display the cards in alphabetical order of file names (1.bmp, 2.bmp, 3.bmp, ...).
- Width of card = 49, Height of card = 72
- When the 'Reset' button is pressed, the front side of the 52 cards is displayed.
- When the '1 Step' button is pressed, flip the cards multiple of next step(multiple of 2, multiple of 3, ...).
  - This process can be repeated up to multiple of 52.
- When the 'Final' button is pressed, display the final result. (After flipping from multiple of 2 to multiple of 52)

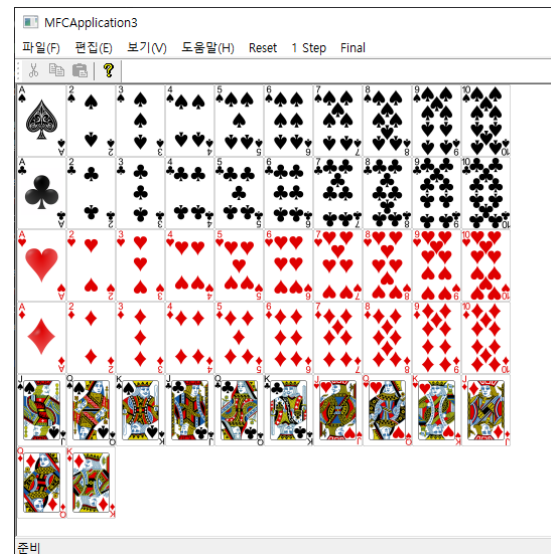


Figure 7 Display Layout

[Example]



Initial state



When 'Reset' button is pressed



After flip the cards multiple of 2



After flip the cards multiple of 3

...



**Figure 8 Final Result**

If you have questions about Assignment 3, contact each assignment below.

<...> [whiteblack4@kw.ac.kr](mailto:whiteblack4@kw.ac.kr) (최한솔, 비마관 201호)

<...> [kjhsbs@gmail.com](mailto:kjhsbs@gmail.com) (김지훈, 비마관 201호)