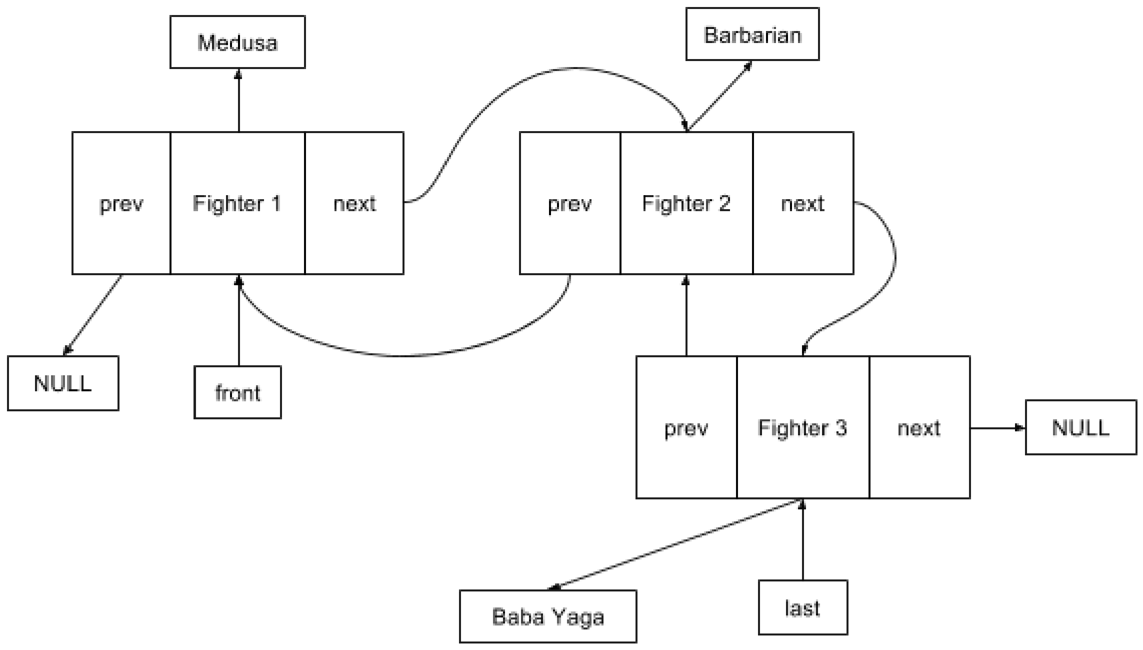
Chermaine Cheang
Assignment 4 Design

Team lineup structure
In order to store lineup of fighters, I will use dynamic queue structures because the first fighter enter the list will be the first fighter fighting. Since queue uses the basis of first-come first-serve, queue structures will be appropriate for storing the lineup. The structures of my lineup will look like the diagram below.



From Module 2, my QueueNode struct was implemented to hold a char data member, for the purposes of this assignment, I will change the char data member to a creature pointer, so that each node now holds a pointer to a creature, in which the creature can be a Barbarian, Blue Men, Baba Yaga, Harry Potter or Medusa. Hence, after instantiated the appropriate creature, I will have to call the add function of Queue class to have the address of this creature stored as a new node in my doubly linked queue.

Keep track of who is who
Since users are allow to create 5 Barbarians or 3 Harry Potter's for their lineup, I will need to have an ID variable in each class to differentiate them. Thus, I will need accessor and mutator functions for this variable, and these functions will be located in the parent Creature class since all other creatures inherit from this class and these functions should remain the same for each class. In addition, I will need to have a constructor that takes a string parameter, which will be the ID, for each class. Again, I can have this constructor in the parent Creature class and all derived class can inherit this constructor. As a result, I will only need to modify Creature class to add these functions.
Winner back to lineup

First and foremost, the winner for each round of combat will be able to go back to its team lineup. Since the lineup is a queue structure, I will add the winner to the back of the queue. That is, if the current player is the winner and it is originally from team 1, I will add that player back to team 1 by calling the Queue class add function.

When the winner is back in line, all their special ability will be reset. That is, if the player is Harry Potter, his death count will be reset to 0, so that he can use his hogwart's ability for the next combat, and check if his current strength point is more than 10, if his strength points is more than 10, then his strength points will be reset to 10 which is his initial strength points. If the player is Baba Yaga and if her strength is currently more than her starting strength points, her strength points will be reset to the initial value which is 12. If the player is Blue Men, his number of defending dice will be reset to 3 to void any Mob effect he previously encountered.

```
void resetAbility(Creature * c1)
{
        Get c1's name - Medusa, Harry Potter, Baba Yaga, Barbarian or Blue Men
        If name == "Baba Yaga"
                If strength > 12
                        setStrength to 12
        If name == "Blue Men"
                If defPoint < 3
                        setDefPoint to 3
        If name == "Harry Potter"
                If deathCount > 0
                        setDeathCount to 0
}
```

After resetting the special abilities for relevant fighters, I will check the fighter current strength points. If the fighter does not have maximum strength points, then he/she has sustained some damage from his/her previous fight. Then he/she will be able to roll a 10-sided die to determine his/her recovery percentage. If a 2 is rolled, they will recover 20% of its current strength. If a 10 is rolled, they will recover fully. That is, if the current player is Barbarian with a current strength of 8, he will be able to roll the die. If he rolled a 4, then his recovery points will be 40% of 8 which is 3 (rounded down). Add this value to his current strength and he now have a strength point of 8 + 3 = 11. If the total strength points of the player after adding the recovery points will exceed their maximum strength points, their strength points will be set to maximum to avoid having strength beyond their limitation. For example, if Harry Potter current has 8 strength points, and he to be received a total of 6 points from recovery, then his total strength will be 8 + 6 = 14. However, because his initial strength point is 10, his strength will be set to 10 instead of 14. Therefore, I will have a recovery function in my parent Creature class, since all creatures will get the same recovery.

```
int randRecover()
{
```

```
                int temp = rand()% 10 + 1;
                int recoverPoint = getStrength() * (temp / 100);
                return recoverPoint;
        }
```

In my driver function, I will have an addition function that will keep take care of all recovery process. This function will only be called after checking if the winner fighter received damage from previous combat.

```
        void recover(Creature* c1)
        {
                Get recoverPoint by calling randRecover()
                Get creature's name - Medusa, Harry Potter, Baba Yaga, Barbarian, or BlueMen
                Switch (creature's name)
                {
                        Case "Barbarian":
                        Case "Baba Yaga":
                        Case "Blue Men": if (recoverPoint + current strength > 12 )
                                                        setStrength to 12
                                                   Else
                                                        setStrength to (recoverPoint + currentStrength)
                                                   break;
                        Case "Medusa": if (recoverPoint + current strength > 8)
                                                        setStrength to 8
                                                 else
                                                        setStrength to recoverPoint + current strength
                                                 break;
                        Case "Harry Potter": if (recoverPoint + currentStrength > 10)
                                                        setStrength to 10
                                                     Else
                                                        setStrength to recoverPoint + currentStrength
                                                     break;
                }
        }
```

<u>Loser to loser pile</u>

The loser from each round of combat will be added to the loser pile container, in this case, since the loser for the first round will be printed last, this will have a stack structure, hence Stack class from module 2 will be used with some modification. Instead of having a char data member, it will now have a pointer to creature variable, and its constructor will now be accepting a pointer to creature as parameter. Loser from each round will be added to the stack using the Stack class add function. At the end of the tournament, an option will be provided to display the contents of the loser pile. The contents of the loser pile will be displayed using the remove function from Stack class. Hence, in my main, I will have

```
Stack loser;
...
...        once a creature loses the battle
loser.add(creature);
...
...        at the end of the tournament

Ask user if they want to print out losers?
If yes
        while (loser is not empty)
                Print out loser.remove()
```

Determine overall winner

The fight between two teams will continue until one of the team has no fighters left to fight. That is if both team started out with 5 fighters, after rounds of combat, if team 1 has 2 remaining fighters and team 2 has none, this is when the tournament ends, and team 1 won the tournament.

```
If (team1.isEmpty())
        Cout << Team 2 won the tournament! << endl;
Else
        Cout << Team 1 won the tournament << endl;
```

Determine top three finishers

The top three finishers will then be determined according to the following conditions:

- If the winning team has only one fighter left, then this fighter will be the champion. The last loser in the loser pile will be the runner-up, and the second last loser will be the second runner-up.
- If the winning team has more than one fighters left, then I will engage these fighters into battles even though they are from the same team. The fighter who won the battle will be returned to the lineup and gets to recover and wait for the next round until only one fighter remaining in the team lineup. The fighter who lost the battle will be added to the loser pile. For example, if the winning team has two fighters left, then these two fighters will go against each other, and the winner will be returned to the lineup while the loser added to the loser pile. Then, the first, second, and third place finishers can be determined using the condition above.

```
        if (team1 is not empty)
                Cout << Team 1 won the tournament! << endl;
                While (team1 fighter's number > 1)
                        Run tournament between fighters in team1
                If (team1 only has one fighter left)
                    first = team1.remove()
                        second = loser.remove()
                        third = loser.remove()
                        print123(first, second, third)

        Else
                Cout << Team 2 won the tournament << endl;
                While (team2 fighter’s number > 1)
                        Run tournament between fighters in team2
                If (team2 only has one fighter left)
                        First = team1.remove()
                        Second =  loser.remove()
                        Third = loser.remove()
                        print123(first, second, third)
```

A function that will be used to print out the top three finishers. This function will receive three pointers to creature as parameters, and will display the first, second and third place finishers according to the sequence of parameters passed in.

```
void print123(Creature* first, Creature* second, Creature* third)
{
    cout << “1st -” << first ->getName() << “ ” << first->getID() << endl;
    Cout << “2nd -” << second->getName() << “ ” << second->getID() << endl;
    Cout << “3rd -” << third->getName() << “ ” << third->getID() << endl;
}
```

Run the tournament

The tournament will start once both teams have their fighters ready for combat. Fighters from each team will be called out using the quece class remove function. The two pointers that will be returned from the remove function will then be passed into startGame function from Assignment 3. Before the combat begins, I will display which type of Creatures will be fighting from each team, and after the combat, I will display who won the combat. The tournament will ends once a team has no available fighter left.

Keep track of which team a fighter is from

To keep track of which team a fighter is from, I will add a variable to the parent Creature class that will store the team information for each creature object instantiated. This way, I will be able to print out which team a creature is from at the end of the tournament.

The main drive

First, I will prompt the user for number of fighters for each team, and check if the user entered number is valid. User input must be numeric and greater than 0. Then, I will ask user to create the lineup for the each team by using the createTeam function describe below.

```
cout << "Enter number of fighters for both players. " << endl;
cin >> number;
```

To avoid repetition of code, I will use a function to create creatures that user specifies. This function will take a Queue parameter which will indicate which team is calling this function, and an integer parameter which will indicate how many creatures to create for each team. A for loop will be used to iterate through the loop x number of times to create x number of fighters and add this fighter into the team queue structure using add function.

```
void createTeam(Queue team, int number)
{
    for (int i=0; i < number; i++)
    {
        Display types of Creatures using a menu and ask user to select
        Prompt user for Creature's name
        team.add(createCreature(team, type, name));
    }
}
```

I will reuse the createCreature function from assignment 3 but with some modification. This function will take 3 parameters, one for team number, one for user's choice of creature, and one for name of the creature. The parameter for team number and name of creature will then be passed into each creature constructor respectively to create that creature with name and team information provided.

```
Creature* createCreature(int team, int choice, string name)
{
    Creature* temp;
    Switch (choice)
    {
        Case 1: //create Medusa
            Temp = new Medusa(team, name);
            Break;
        …
    }
    Return temp;
}
```

After both team are done with creating their fighters, the tournament will be begin with the first fighters from each team having a combat between one another. To get fighter from each team lineup, the remove function will be used, and temp fighter variable will be used to store the pointer returned from the remove function. The combat will continue until one fighter is defeated that is, when one fighter has strength equals 0 or less than 0. The startGame function from Assignment 3 will be used for facilitating the combat between two fighters. However, instead of naming the function startGame, I will change it to combatBetwFighters to indicate that this is a combat between two fighters.

A function will also be used to facilitate the entire tournament. This function will have two parameters, which will be the team lineups. This function will call the appropriate fighters into combat by calling the combatBetwFighters function. Before starting the combat between two fighters, this function will display which creatures are fighting using displayFighter function, and after each round of combat, this function will display who won the combat by calling the displayCombatWinner function. After determining who won the combat, the winning fighter will have the chance to recover by calling the recovery function which will call its randRecover function from its class. After recovering some percentage of the fighter strengths, resetAbility function will be called if the fighter is either Harry Potter, Baba Yaga or Blue Men to reset their superpower ability before adding the fighter back into its team lineup.

```
void startTournament(Queue team1, Queue team2)
{
    Creature* fighter1 = NULL;
    Creature* fighter2 = NULL;
    while (team1 and team2 is not empty)
    {
        fighter1 = team1.remove()
        fighter2 = team2. remove()
        displayFighter(fighter1, fighter2)
        combatBetwFighters(fighter1, fighter2)
        Determine which fighter won the combat
            displayCombatWinner()
            Call recover() to recover the winning fighter
            Call resetAbility() if fighter is either Blue Men, Baba Yaga, or Harry
            Potter
            Add fighter back to team lineup
            Add defeated fighter to loser stack
    }
}
```

The while loop in startTournament will terminate when one of the team has no fighters left. This will mark the end of the tournament, and the overall winning team and top three finishers will be determined using the method described above. An option will also be provided to user if they wish to see a list of the final order of all fighters.

Pseudocode

```
int creatureType()
{
    Display list of creatures available
    Prompt user for their option
    Validate user's choice
    Return user's choice
}

Creature* createCreature(int team, int choice, string name)
{
    Creature* temp;
    Switch (choice)
    {
        Case 1: //create Medusa
            Temp = new Medusa(team, name);
            Break;
        ...
    }
    Return temp;
}

void createTeam(Queue team, int number)
{
    for (int i=0; i < number; i++)
    {
        Display types of Creatures using a menu and ask user to select
        Prompt user for Creature's name
        team.add(createCreature(team, type, name));
    }
}

void startTournament(Queue team1, Queue team2)
{
    Creature* fighter1 = NULL;
    Creature* fighter2 = NULL;
    while (team1 and team2 is not empty)
    {
        fighter1 = team1.remove()
        fighter2 = team2. remove()
        displayFighter(fighter1, fighter2)
        combatBetwFighters(fighter1, fighter2)
        Determine which fighter won the combat
```

```
                    displayCombatWinner()
                    Call recover() to recover the winning fighter
                    Call resetAbility() if fighter is either Blue Men, Baba Yaga, or Harry
                    Potter
                    Add fighter back to team lineup
                    Add defeated fighter to loser stack
        }
}

void print123(Creature* first, Creature* second, Creature* third)
{
        cout << “1st -” << first ->getName() << “ ” << first->getID() << endl;
        Cout << “2nd -” << second->getName() << “ ” << second->getID() << endl;
        Cout << “3rd -” << third->getName() << “ ” << third->getID() << endl;
}

int main()
{
        Int numFighters;
        Queue team1;
        Queue team2;
        Stack loser;

        Do {
                Get numFighters from user
                Validate user entered numFighters
        } while (not valid);

        createTeam(team1, numFighters)
        createTeam(team2, numFighters)

        startTournament(team1, team2)

        if (team1 is not empty)
                Cout << Team 1 won the tournament! << endl;
                While (team1 fighter’s number > 1)
                        Run tournament between fighters in team1
                If (team1 only has one fighter left)
                        first = team1.remove()
                        second = loser.remove()
                        third = loser.remove()
                        print123(first, second, third)
```

```
    Else
        Cout << Team 2 won the tournament << endl;
        While (team2 fighter's number > 1)
            Run tournament between fighters in team2
        If (team2 only has one fighter left)
            First = team1.remove()
            Second =  loser.remove()
            Third = loser.remove()
            print123(first, second, third)

    Ask user if they want to see a list of final order
    If yes
        Cout << loser.remove() << endl;
}
```

__Testing plan__

Test Plan	Input value	Drive function	Expected outcome
1	Prompt user for number of fighters for each team and validate user's input. *User enters 0*	validate(int)	Invalid input. Prompt user to enter number of fighters for each team again.
2	Prompt user for number of fighters for each team and validate user's input. *User enters any characters*	validate(int)	Invalid input. Prompt user to enter number of fighters for each team again.
3	Prompt user for number of fighters for each team and validate user's input. *User enters 5*	validate(int)	Input is valid.
4	Display the list of creature types	creatureType()	List of creature is displayed and waiting for user choice

5	User wants to create a Barbarian for team 1 and name this Barbarian b1	creatureType()
		createCreature()
			New Barbarian is created with name b1 and added to team1 queue.

6	User created fighters for team 1	createTeam()
		getNumItem()
			Team 1 has a total of 5 fighters

7	User created fighters for	createTeam()	Team 2 has a total of 5 fighters

	team 2	getNumItem()	
8	Remove the first fighters from both team	remove()	First fighter from both teams remove from their team lineup Each team now has a total of 4 fighters
9	Battle between the two first fighters	combatBetwFighters()	Each round of attack and defend was displayed on screen.
10	Determine who won the battle by checking if their alive is true	startTournament()	Display the fighter who won the battle

11	Recover the winner	recover()
		randRecover()
			The fighter strength is recovered by some percentage depending on the value returned from randRecover()

12	Fighter is Harry Potter. Reset Hogwarts ability	resetAbility()	Reset Harry Potter's death count to 0
13	Fighter is Baba Yaga Reset Soul ability	resetAbility()	Reset Baba Yaga's strength to 12 if Baba Yaga's current strength is more than 12
14	Fighter is Blue Men Reset Mob ability	resetAbility()	Reset Blue Men's defPoint to 3 if Blue Men's current defPoint is less than 3
15	Return fighter to team lineup	add()	Fighter returned to back of team lineup. Team now has 5 fighters
16	Add loser to loser stack	add()	Loser is added to the loser pile
17	Run the tournament until one team has no available fighter left	startTournament()	The while loop in startTournament kept iterating until either team1 has no fighter left or team2 has no fighter left. Exit the loop

18	Team 1 has no available fighter left. Team 2 has one fighter left	main()
		print123()
			Team 2 won the tournament. Displayed the top three finishers.

19	Team 1 has no available figther. Team 2 has 2 fighters left	main()
		combatBetwFighters()
		print123()
			Display attacks and defends points for each round of attack between the two remaining fighters from team2.

			Print out team2 won the tournament Displayed the top three finishers in which the top two will be from team2