Program Overview
This project is a continuation of project 3. Where project 3 calculated the combat between 2 fighters, project 4 calculates the combat between two groups of fighters. The 5 different types of fighters are the same between the projects. After each fight the winner goes to the bottom of his/her stack and the loser goes to a loser stack. Fights continue until one side has lost all of its fighters. Each round won is tallied for both sides and at the end the overall winner is declared. Since there are so many similarities between this project and project 3 the rest of this document will only discuss items that are different between this project and project 3.

Program Design
The only addition to the overall structure of the program is the addition of a struct header file to define the struct Combatants which holds a pointer to a Fighter object and a pointer to another Combatants object. The Fighter class was given an additional constructor to allow for the variables to be defined. With that in place the other classes can correctly inherit from Fighter and as such any methods they use which are identical to those of Fighter were removed. Major changes were done to the Battle class to allow for stacks of fighters and continuous combat rounds.

Fighter Class
A new variable, maxHealth, was added to the Fighter class. A new method, recoup, was added to the Fighter class. Finally, a constructor was added to set all of the Fighter variables based on arguments.

The recoup method determines the amount of health the Fighter has lost, takes a random percentage of the lost health and adds it back to the Fighters health.

Harry Potter Class
The resurrect method was modified to change the maxHealth variable when called so the recoup method will function correctly after Harry Potter is resurrected.

Battle Class
The Battle class now has 3 Combatants pointers instead of 2 Fighter pointers, it also has p1Wins, p2Wins, and size integer variables. All of the existing methods were modified to account for the use of stacks of Fighters. The following methods were added: addFront, addBack, removeFront, delStack.

The menu method has been modified to ask for the number of fighters for each team and then loop through fighter generation for each team before calling the fight method then deleting the stacks and asking the player if they would like to play again.

The fight method was modified to have the front of the p1 and p2 stacks attack each other until one has died then display those results, move the loser to the lose stack, and move the winner to the bottom of his/her stack. This will loop until one of the stacks is empty. When that happens, the final results are displayed and a list of all of the fighters who lost their battle is also displayed.

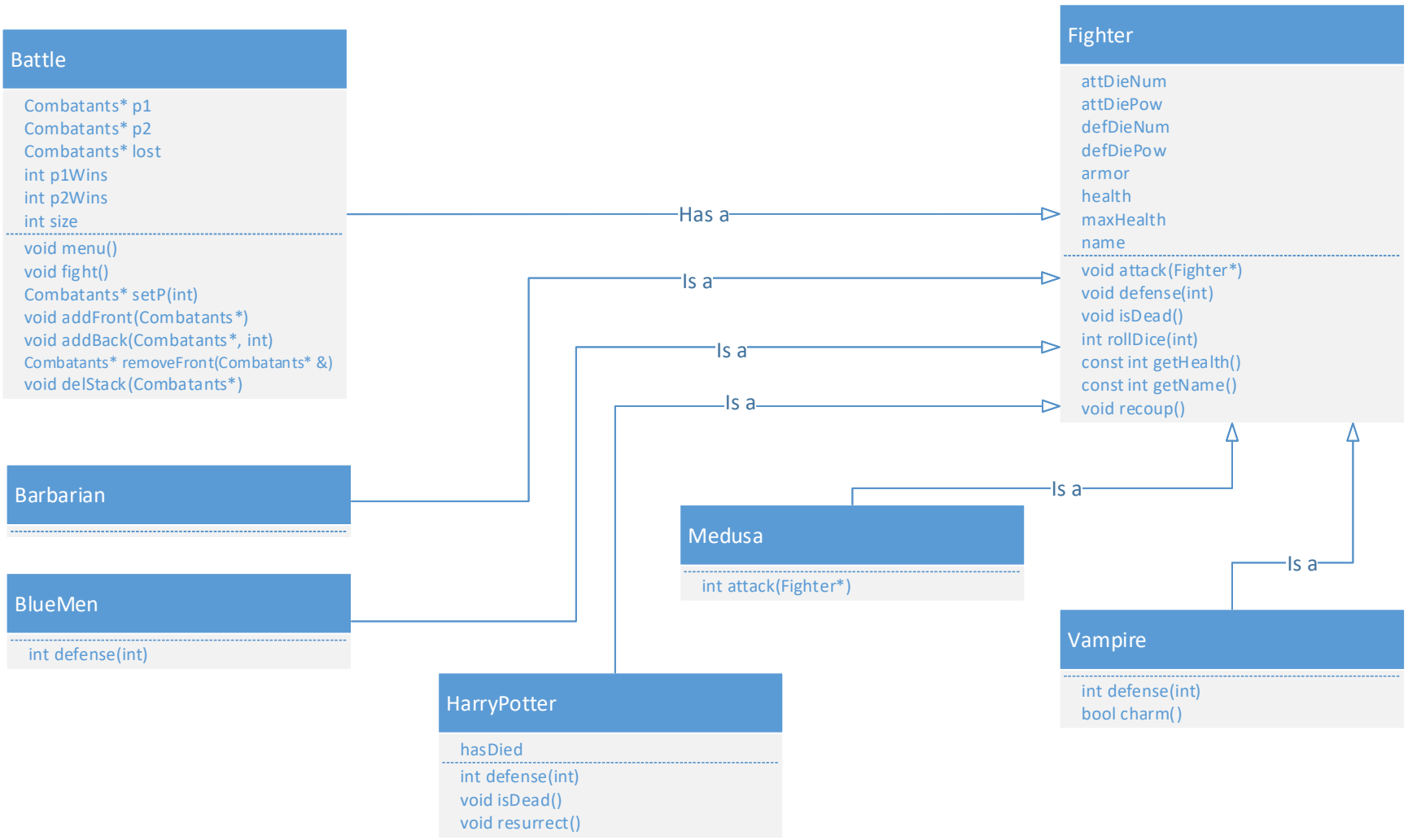
The setP method was modified to create and return Combatants pointers with the correct Fighter class in the Combatants struct.

The addFront method is a new method which adds the Combatants pointer argument to the front of the lost stack.

The addBack method is a new method which adds the Combatants pointer argument to the back of either the p1 or p2 stacks as determined by the int argument.

The removeFront method is a new method which remove the first Combatants struct from the stack pointed to by the Combatants pointer argument and returns it.

The delStack method is a new method which goes through a stack of Combatants structs and deletes all of them.


Battle
Combatants* p1
Combatants* p2
Combatants* lost
int p1Wins
int p2Wins
int size
void menu()
void fight()
Combatants* setP(int)
void addFront(Combatants*)
void addBack(Combatants*, int)
Combatants* removeFront(Combatants* &)
void delStack(Combatants*)
Has a
Fighter
attDieNum
attDiePow
defDieNum
defDiePow
armor
health
maxHealth
name
void attack(Fighter*)
void defense(int)
void isDead()
int rollDice(int)
const int getHealth()
const int getName()
void recoup()
Is a
Barbarian
Is a
BlueMen
int defense(int)
Is a
HarryPotter
hasDied
int defense(int)
void isDead()
void resurrect()
Is a
Medusa
int attack(Fighter*)
Is a
Vampire
int defense(int)
bool charm()

Test

The testing for this program, due to the number of combinations, will not be an exhaustive list of all possibilities but will instead focus on error handling, edge cases, and a range of normal cases. The first thing to be tested is error handling.

Error Handling Tests

Requested input	Expected values	Actual input	Program response
Number of fighters per team	0 < x < 11	0	"User input invalid, please try again."
Number of fighters per team	0 < x < 11	11	"User input invalid, please try again."
Number of fighters per team	0 < x < 11	abc	"User input invalid, please try again."
Choose a fighter	0 < x < 6	0	"User input invalid, please try again."
Choose a fighter	0 < x < 6	6	"User input invalid, please try again."
Choose a fighter	0 < x < 6	abc	"User input invalid, please try again."

Edge Case Tests

Requested input	Expected values	Actual input	Program response
Number of fighters per team	0 < x < 11	1, Barbarian (T1) vs Medusa (T2)	Team 1 won.
Number of fighters per team	0 < x < 11	10, 2 of each Fighter per team	Team 1 won.

Normal Case Tests

Requested input	Expected values	Actual input	Program response
Number of fighters per team	0 < x < 11	2, Barbarian and HP each team.	Team 1 won.
Number of fighters per team	0 < x < 11	3, Barb, HP, and Medusa each team.	Team 2 won.
Number of fighters per team	0 < x < 11	4, Barb, HP, and Medusa each team.	Team 2 won.
Number of fighters per team	0 < x < 11	5, one of each fighter per team.	Team 1 won.

Reflections

This program was more time intensive than I initially anticipated. As I found out it was slightly more complex than just taking Program 3 and adding stacks. I had some issues with getting the stacks to function correctly. More specifically my removeFront method in my Battle class wasn't actually removing the first Combatants struct from the argument it was given. After some troubleshooting on this I realized that I was passing the argument by value instead of by reference. Changing that resolved the issue. I do still have an issue with memory leaks for this program. Running valgrind on the program shows 2 blocks definitely lost and 4 indirectly lost. I don't believe the issue is directly related to the Fighter objects because the number of leaks doesn't change when the program is run with a different number of Fighters. The number of leaks doesn't match up with the number of Combatant structs either. The printout makes some reference to the setP function but modifying that did not result in a change in the errors. If the grader sees why there are memory leaks for this program I would really like to know so I can deal with this in the future if it comes up.