Assignment 4 – Fighting Game Tournament

Understanding of the Problem -
Expand the existing fighting game to include tournament functionality, by using an appropriate data structure. Have the two fighters at the top of the lineup fight each other. The winner will be sent to the back of the lineup, and the loser will be knocked out of the tournament. Create a scoring system, and calculate the three most useful fighters at the end of the tournament.

Design of a Possible Solution -

```
struct fighter
{
creature * creature_pointer;
string player_name;
}

vector <fighter> player1_team;
<<player1, enter name;
>>player1;
Each player gets to pick a number of creatures.
while (add more creatures == true)
{

What type of creature do you want?
Choose 1-8;

if (choice == 1 (elf))
{
fighter new_fighter;
elf* elf_pointer
new_fighter.creature_pointer = elf_pointer;
}

if (choice == 2 (troll))
{ … same thing…}

new_fighter.player_name = player1;
player1_team.pushback(new_fighter);
<< Add more creatures?
>>y_or_n;
}

For player 2, while (#_added < #_player1_added)
{ … do the same thing…}
do  r(player1_team[1], player2_team[1]);

{
encounte
if (player2_team[1]>get_strength() > 0)
{
encounter(player2_team[1], player1_team[1]
}
}(while player1_team[1]>get_strength() > 0 &&
player2_team[1]>get_strength() > 0)

if (player1_team[1]>get_strength() <= 0)
{
loser_list.pushback(player1_team[1]);
player1_team.erase(player1_team.begin(),
player1_team.begin()+1)
player2_team.pushback(player2_team[1])
player2_team.erase(player2_team.begin(),
player2_team.begin()+1)
}
```

This was the original design I submitted to my TA, Jan "Uli" Ulrich Bartels. He then recommended that I used a linked list structure to hold my teams. I used little of the actual design I proposed. After switching my focus to using a linked list, I reformulated my design:
Still using struct fighter;
Now with the additional fields of
int player number (1 or 2);
fighter pointer fighter_ptr;

Create a node, create a second node linked to the first, then a third until the number of fighters that the

players specified is reached.
Create a loop, by pointing the last link to the first node.
In creating each node, allow the player to pick from numbers 1-7, which represent different creatures.
Use a switch to create a new creature of the correct type, and add it to the node.

Adapt the “encounter” function from Assignments 3-4 (renamed “strike”) for fighter pointers.
Create a new “encounter” function that has one fighter strike at the other until one of them has 0 strength remaining.
When a fighter “passes out” and has 0 strength, remove their link in the chain, but do not delete the node. Instead, send it to a loser's list vector, so that the losers can be printed in order they were eliminated (decided not to print this out, in the end)

Keep having the next fighters in the list pair off against each other, eliminating one fighter each encounter, until one team has 0 fighters left. That team loses, the other team wins.

After discussing my design with Uli some more, I learned that I needed to find some way to calculate the individual fighters were most useful in the tournament. I decided to base this on most damage done.

Create a new variable for fighters that holds the cumulative damage done over their multiple battles.
~~Sort a vector containing all of the fighters in order of most damage done.~~
Create a function that has 3 pointers for 1, 2, 3 places. Iterate through the vector, comparing the current fighter's total damage score.

If current > first place,
first place is now current
first place moves to second place
second place to third
if current > second place
… same, but for second place.

Test Plan:
Make teams of various sizes (2, 99, several in between), with all different creatures and with all the same creatures.
Play the tournament several times, quitting at the first opportunity, and playing multiple tournaments without quitting.
Try to select a number that does not correspond to a creature. (caught by input checker)
Try to select a team > 100, or < 2. (caught by input checker)
Try to enter various player names including numbers and characters (works fine. Player can call themselves whatever they want, including $nuff@a1uff@gu5, if they so choose)

Print out the list of losers and the list of fighters still standing and count to make sure that the numbers equal the total number of fighters.
Print out the data from the damage_sort function, as it calculates, to make sure it is working as expected

Reflection -
This final project was quite a journey. I felt like my direction changed several times. I think it will be most helpful to discuss my process in phases, since that is how I feel I worked my way through this project.

Phase 1: I created and submitted my design, which I was confident about. I was told that it would create a scope problem, which was something that I had already run up against in Assignment 4, and was happy to avoid. I was told to use a linked list as my main data structure for the assignment.

Phase 2: I reviewed my lab on linked lists, and created a linked list of struct pointers. I felt like this part of the process went fairly smoothly. As I was still getting used to the way linked lists behave, I experimented with passing different pointers in and out of my functions. I ended up deciding to return the last item of my linked list from the funciton that creates the list instead of the head, which was what I had originally intended (and why the variables are called head_1, head_2). This is because I quickly discovered that if I wanted to remove the loser of the battle from the list, I needed to pass the battling function both the fighter and the pointer before the fighter. Thus, I passed the function head_1, and head_1->link, and head_1->link did the battling. I wanted to make sure that the fighters fought in the order they were chosen, in case the player had a strategy in mind.
I got much of my program written the night after I found out that I needed to use a linked list, because I got “in the zone” and just wanted to keep working.

Phase 3: I received an email from Uli stating that he had just found out that we were not supposed to use linked lists for the final project, but that, if we were already so far in that we did not want to change our data structure, we could write an implementation file for the linked list, and would not lose points. I was very disheartened by this, since I would have been happy to use any other data structure, but had been told specifically to use the linked list, and I already had a working program that was very nearly finished, using the linked list structure. I looked into the other data structures, and reviewed my program, and decided to create the implementation file for my program.

I created the implementation file inside the creature.cpp and creature.h files, because, in order to declare the struct for the linked list, I needed to have access to the class creature. I left the function for creating a team outside of the creature.cpp file, and put it in the main program file (tournament) because it needed to have access to all the different creature sub-classes in order to populate each node. It is clearly labeled as a part of the implementation both in it's declaration and in it's definition.

I changed all of the references to links to accessors for links, so that I was not accessing my linked list “willy nilly” and also expelled fighters from the list using an accessor. I tried several times to create a function that initialized an individual fighter, in addition to the function that initialized a team, but for some reason, both assigning original values to the struct, and creating a new fighter from functions declared in the implementation file caused run-time errors. After several different versions of the function, which gave me no positive results, and with no clue where the run-time errors were coming from, except that the linked list was seemingly never being created, I gave up, and decided that I would set original values for the struct only within the single, team initializing, function.

Phase 4: I re-read the instructions to see whether I meeting all the requirements, and, as seems to be extremely common for instructions in this course, there were some requirements that left me completely confused as to whether I was meeting the requirement, as many of the things in the various assignment instructions have seemed like suggestions and examples, and the actual requirments have been vague and unclear. I emailed Uli to ask how I was supposed to declare three winning fighters, that

may not be on the winning team, since in my view, the winner would be the player and all creatures that survive the tournament are the winning creatures. Uli explained that I need to determine the three most “useful” creatures, in any way that I decided to do so. I decided to implement a total number of “wins” or battles won, and then shortly afterward changed my mind and decided instead to calculate total damage done, over all battles fought, whether won or lost.

First, I tried adding a new field to the struct, but doing so created run-time errors, again. I was not able to figure out where they were coming from. After deciding to try a new approach, I added get_wins() and set_wins() to my creature class, and added this turn's total number of damage done to the overall total damage done, so that at the end, I could see which creatures had done the most damage. I called the first, second and third standings the VIPs of the tournament, instead of the winners, since the winner was the player.

I created a vector that contained both the loser_list vector, as well as all the fighters still standing on the winning team. I had originally printed the loser_list, showing the order in which creatures were defeated, in an attempt to satisfy the requirement for a “standing” which I did not understand. After having it clarified by Uli, I chose not to print the loser_list, as it seemed like there was enough information being presented at the end of the tournament without it. However, if it were useful, at a later time, to reveal the order in which fighters were eliminated, the information is still accessible.

I tried to use the algorithm “sort” from the standard template library, to sort the creatures by amount of damage done. On cplusplus.com, I reviewed the implementation, and wrote a function that defined what I wanted to be compared when sorting: namely, the get_wins() number, from the class creature, in the struct fighter. I ran the program several times. The sort algorithm changed some of the numbers around. It generally put some of the high numbers at the front of the vector. One time it correctly sorted 8 numbers into descending order, however, after further testing this appeared to be a coincidence. I was not able to manipulate the function to get it to sort all of the numbers into the correct order reliably.

My next strategy was to attempt to overload the comparison operators in the class creature. I tried to extrapolate from previous experience overloading and from my notes, and wrote a member function to overload the '<' operator, which took one argument. The compiler said that the function to overload '<' must take two arguments. I re-wrote the function to accept two arguments, at which point, the compiler said that the function to overload '<' must take one argument. I then googled how to overload the '<' operator, and read several lists of recommended operators to overload, and noted that the '<' operator was not among them.

My third strategy was to use the vector sorting function that I had created for a lab, and saved for later use. Although it compiled and ran as expected in it's own program, my own sort function would not modify the vector in tournament. I tried printing out the vector inside the function, to see if the vector was being reverted when returned. The vector was not being modified inside the function where the modification ought to have been happening.

My fourth and final strategy was to stop trying to order the entire vector, which seemed, originally, to be the quickest and easiest way to get what I wanted. I decided to go after only the information that I needed, which was the top three highest scores in the tournament. I devised a function that compared each node in the vector to the current scores in first, second and third place. If it was higher than any of the existing scores, the existing score was replaced by the current node, and the rest of the score board was adjusted relatively. I placed my print statements inside the loop, which turned out to be a happy accident, as, when I ran the program, I was able to see the function work at each iteration, and was

thrilled to find out that it worked as expected and reliably produced the three top damage-dealing fighters.

I am happy to be done with this assignment, as the last step was, as you have seen, excruciatingly tedious. I think that my final product is a good one, and meets the requirements effectively. I am proud, looking at this program, and seeing how far I have come toward proficiency in C++. I intend to continue to use it and practice, with a focus on working more effectively with the STL. I also intend to attain and learn a new IDE, which is running C++ 11 so that I can take advantage of the new advances that have been mentioned in this course.