Design Document: Polymorphism (Tournament)
CS 162, Assignment 4
Collin James

Initial Design
main():

- create and instantiate 2 team pointers
- create Tournament pointer
- user 1 inputs number of creatures per team (intakeInt())
- user 1 enters that number of creature types (loop) (mainMenu() variant, intakeInt ())
    - call addCreature(type) helper function (case switch with types), which calls team::addMember
- user 2 enters that number of creature types (loop)
    - call addCreature(type) helper function (case switch with types)
- add 2 teams to tournament object by instantiation
- call Tournament::doTourney() to do the tournament
- call Tournament::printStandings()
- prompt user to enter yes/no for print winner (intakeString(), while loop)
    - if yes, call tournament::printStandings()
        - delete pointers, end program
    - if no, delete pointers, end program

Tournament:

- pass in two teams on instantiation
- doTourney()
    - while team1 is active and team2 is active
        - pick topMember() of team
        - doRound(team1->getTopMember(), team2->getTopMember())
- doRound(creature1, creature2)
    - while !isDead and !isDead
        - damage is result of creature1 attacking creature2
            - if damage greater than 0, add points for correct team with computePoints()
            - if damage > 0, team2->addPoints(computePoints(damage))
    - run Team::moveFighters() on both teams
    - move dead creature to standings
- void moveToStandings()

        - move a dead creature to standings stack
- int computePoints(int damage):
    - return damage • defender strength rating ÷ attacker strength rating
        - strength ratings: Goblin = 1, Barbarian = 2, Shadow = 3, Reptile = 4, BlueMen = 5
    - if damage is < 0, means fight is won
        - return 10 points for winner of fight
- printWinner()
    - print top 3 fighters
        - run Stack getTop() 3 times and print to screen
        - move top 3 to topThree queue
    - print winning team info
        - compare and print appropriate team winner
        - print points for both teams
- printStandings()
    - if topThree getFront()
        - print topThree (removeFront 3 times)
        - print standings stack (loop until…)
    - otherwise (top 3 not determined)
        - print standings stack (loop until…)

Team:

- Create empty queue and stack on instantiation; set points & activeMembers to 0
- addMember(Creature *)
    - add it to the members queue
    - add 1 to activeMembers
- getFrontMember()
    - return the result of members->getFront() or NULL if no getActiveMembers ()
- getActiveMembers()
    - return activeMembers integer
- moveFighters()
    - run members->getFront->isDead()
        - if dead, removeFront() from members and addTop to losers stack
        - if not dead, Creature::restoreDamage() on front creature, removeFront(), and addBack() to members queue
- addPoints(int points)
    - add points to this->points

- int getPoints()
  - return points
- resetTeam()
  - if creatures in losers stack, add them back to queue
  - call Creature::revive() on each creature

Creature
- Same as Assignment 3 creatures, except
  - Add restoreDamage() to base Creature class

Revised Design

main():
- create and instantiate 2 team pointers
- create Tournament pointer
- user 1 inputs number of creatures per team (intakeInt())
- user 1 enters that number of creature types (loop) (mainMenu() variant, intakeInt())
  - call addCreature(type) helper function (case switch with types), which calls team::addMember
- user 2 enters that number of creature types (loop)
  - call addCreature(type) helper function (case switch with types)
- add 2 teams to tournament object by instantiation
- do the tournament
- print the **winner**
- prompt user to enter yes/no for print **standings** (intakeString(), while loop)
  - if yes, call tournament::printStandings()
    - delete pointers, end program
  - if no, delete pointers, end program

Tournament:
- pass in two teams **and number of fighters** on instantiation; **create empty Stacks for standings and top 3; create empty IntStacks for team names and top 3 names; create string of names of creatures**
- doTourney()
  - while team1 is active and team2 is active
    - pick topMember() of team

        - doRound(team1->getTopMember(), team2->getTopMember())
- doRound(creature1, creature2)
    - while **both creatures are alive**
        - damage is result of creature1 attacking creature2
            - if damage greater than 0, add points for correct team with computePoints()
            - if a creature is dead, **add kill bonus points**
    - run Team::moveFighters() on both teams
    - move dead creature to standings
- void moveToStandings()
    - move a dead creature to standings stack
- int computePoints(int damage):
    - return damage • defender strength rating ÷ attacker strength rating; **be sure to cast as double while doing division and multiplication**
        - strength ratings: Goblin = 1, Barbarian = 2, Shadow = 3, Reptile = 4, BlueMen = 5
    - ~~if damage is < 0, means fight is won~~
        - ~~return **50** points for winner of fight~~
- printWinner()
    - **finish moving all members of team to standings Stack (also add team numbers to appropriate IntStack)**
    - print top 3 fighters (**or top # of fighters if less than 3**) **and teams**
        - run Stack getTop() 3 times and print **type and team** to screen
        - move top 3 to topThree **Stack (and corresponding name Stack)**
        - **move top 3 back to top of standings Stack (and corresponding name Stack)**
    - print winning team info
        - compare and print appropriate team winner
        - print points for both teams
- printStandings()
    - if **a creature is at top of standings**
        - ~~print topThree (removeFront 3 times)~~
        - print standings stack **by removing one by one** (loop until **NULL is returned and standings is empty**)
- **finishStandings()**
    - **move all remaining members of Teams to standings Stack**

Team:

- Create empty queue and stack on instantiation; set points & activeMembers to 0
- addMember(Creature *)
  - add it to the members queue
  - add 1 to active_members
- getFrontMember()
  - return the result of members->getFront() **(may be NULL if no creatures in Queue)**
- **releaseFrontMember(**
  - **return the result of members->removeFront() and decrement active_members**
- getActiveMembers()
  - return activeMembers integer
- moveFighters()
  - run members->getFront->isDead()
    - if dead, removeFront() from members and add() to losers stack
    - if not dead, Creature::**heal**() on front creature, removeFront(), and addBack() to members queue
- addPoints(int points)
  - add points to this->points
  - **return current points**
- int getPoints()
  - return **current** points
- resetTeam()
  - if creatures in losers stack, add them back to queue
  - call Creature::revive() on each creature
    - **requires removing them to a temporary queue and then back to members queue**
- **getNumber()**
  - **return the team number (i.e. 1 for "team 1")**

Creature

- Same as Assignment 3 creatures, except
  - Add **heal**() to base Creature class
  - **Add kill() to base Creature class (for testing)**

Reflection

The design came together fairly easily for this assignment. I knew that I would have to alter my Stacks and Queues to take Creature pointers instead of integers (and that my Queue would be circular), and I knew that altering Creature to heal() would be easy once I figured out an algorithm. The algorithm that I came up with was simple but did the trick. The main part of the design was creating classes to handle the actual tournament and team concepts. As you can see in my design documents, I decided that Tournament was my top class, which contained two Teams and a Stack of Creatures. Teams contained a Queues and Stack of Creatures. The concept was that Team would control its Queue of Creatures and Tournament would orchestrate the fighting and point distribution of its Teams. My basic design of these classes stayed the same, although I added some convenience functions and logic where necessary, which can be seen in the design notes.

A problem arose late in coding when I realized that I had forgotten to figure in team names when printing the standings and winners. My idea was to create companion Stacks for my members in Team. This Stack (called IntStack and nearly identical to the lab 6 Stack) just holds integers instead of Creatures and is a mirror of the member Stack, and is updated similarly to the other Stack. I would like to have instead implemented a Stack (and Queue) that held both Creature pointers and team numbers, but it would have been too much work late in the game. My fix worked fine despite its inelegance.

As noted in the testing docs, I had a problem with segmentation faults early on in testing Tournament. Using gdb and (mostly) my own cout statements, I narrowed it down to my trying to delete pointers that were pointing to already-deleted memory. I found that I was deleting the two Team pointers in the Tournament deconstructor and then again in main(); once I stopped deleting in main(), the problem stopped. I decided it was best to let the Tournament deconstructor take care of the deletion of these pointers.

All told, it was a fun and pretty satisfying assignment. I was able to reuse a lot of code that I had created earlier in the semester, particularly from Assignment 3, Lab 6 and 8, and Assignment 2. That sped up the process considerably, and I am glad that I took the time to make sure the earlier code worked solidly when doing previous assignments.

Initial Class Hierarchy: Inheritance
CS 162, Assignment 4
Collin James

Tournament

```
Team* one, Team* two
Stack* standings
Queue* topThree
enum Strength={Goblin, Barbarian...}
string names
```

```
Tournament(Team* one, Team* two)
~Tournament()
void doTourney()
void doRound(Creature* one, Creature* two)
int computePoints(int damage)
void moveToStandings(Creature* dead)
void printWinner()
void printStandings()
```

(Tournament has-a Team)

Team

```
int points, active_members
Queue* members
Stack* losers
```

```
Team()
~Team()
void addMember(Creature*)
void moveFighter(Creature*)
int addPoints(int)
Creature* getTopMember()
void resetTeam()
int getActiveMembers()
```

(Team has-a Stack)

(Team has-a Queue)

(Tournament has-a Stack and a Queue)

Queue

```
struct Queuenode
   - Creature* value
   - Queuenode* ahead
   - Queuenode* behind
Queuenode* front
Queuenode* back
```

```
Queue(Creature* val)
~Queue()
bool addBack(Creature*)
Creature* removeFront()
Creature* getFront() const
void displayQueue()
```

(Queue has-a Creature)

Stack

```
struct Stacknode
   - Creature* value
   - Stacknode* next
Stacknode* head
```

```
Stack(])
~Stack()
void add(Creature*)
Creature* remove()
Creature* get() const
void displayStack()
```

(Stack has-a Creature)

Creature

```
DieVal attack, defense (struct)
  - int num, int sides
int armor,
double strength, orig_strength
bool is_dead,
  attack_halved,
  is_knocked_out
CharType type (enum)
```

```
Creature(int, int, int, int, int, double,
CharType)
~Creature() – virtual
int rollDice(DieVal)
double inflictDamage(double, bool)
double computeStrength(double)
double doAttack(Creature* int, bool)
double attack(Creature*) = 0;
double defend(double, CharType, bool) = 0;
bool isDead()
bool isKnockedOut()
double getStrength()
int getRoll(),
CharType getType(),
void listStats(),
void revive()
void heal()
void kill()
```

(Shadow is-a Creature)

Shadow

```
int disappeared
```

```
Shadow() : Creature()
~Shadow()
double attack(Creature*)
double defend(double,
CharType, bool)
```

(Reptile is-a Creature)

(Barbarian is-a Creature)

(Goblin is-a Creature)

(Bluemen is-a Creature)

Reptile

```
(inherit from
Creature)
```

```
Reptile() : Creature()
double attack(Creature*)
double defend(double,
CharType, bool)
```

Barbarian

```
(inherit from
Creature)
```

```
Reptile() : Creature()
double attack(Creature*)
double defend(double,
CharType, bool)
```

Goblin

```
int halved
```

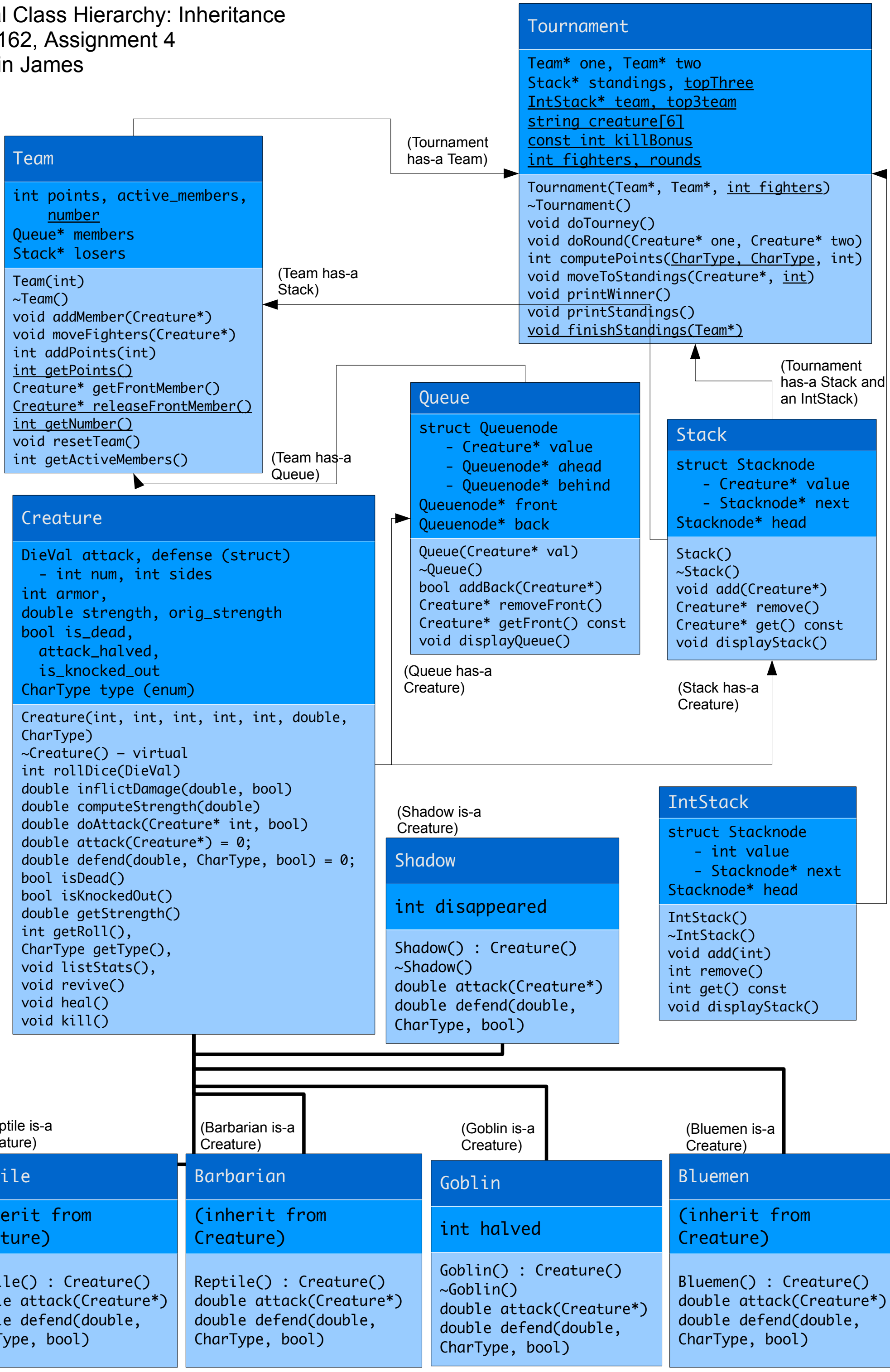
```
Goblin() : Creature()
~Goblin()
double attack(Creature*)
double defend(double,
CharType, bool)
```

Bluemen

```
(inherit from
Creature)
```

```
Bluemen() : Creature()
double attack(Creature*)
double defend(double,
CharType, bool)
```

Final Class Hierarchy: Inheritance
CS 162, Assignment 4
Collin James



Testing Document: Polymorphism (Tournament)
CS 162, Assignment 4
Collin James

I tested early and often and made commits in my version control system whenever code was working as expected. Changes from original plan are listed in **bold**.

Testing Plan

Test Case	Input Values	Driver Function	Expected Outcomes	Observed Outcomes
heal a Creature	n/a	Creature::heal()	strength restored by algorithm: (max strength - current strength) ÷ factor(2 is default)	strength restored by algorithm: (max strength - current strength) ÷ factor(2 is default)
add Creature to Stack and Queue	n/a	add(), addBack()	Creature is added to Queue or Stack	Creature is added to Queue or Stack
add Creature to Team	n/a	addMember()	Creature goes to back of Queue	Creature goes to back of Queue
delete Team	n/a	~Team()	pointers are deleted	pointers are deleted
add points to Team	points	addPoints()	Team points increase by proper amount	Points sometimes increase by 0; problem was caused by trying to do division with ints; fixed by doing static casts to doubles and rounding up with ceil() before sending to addPoints()
add Teams to Tournament	Team pointers	Tournament()	Teams are properly copied to Tournament	Teams are properly copied to Tournament
do a round of fights in Tournament	n/a	doRound()	round continues until one creature dies	round continues until one creature dies
do several rounds in Tournament	n/a	doTourney()	rounds continue until all creatures on one team are dead	rounds continue until all creatures on one team are dead

Test Case	Input Values	Driver Function	Expected Outcomes	Observed Outcomes
print the winner	n/a	printWinner()	prints top 3 fighters and winning team and points	prints top 3 fighters (**and teams; added IntStack class**) and winning team and points (**and losing team points**)
print the standings	n/a	printStandings()	print all fighters in order of last alive to first dead	problem removing from team roster; **created Tournament::finishStandings() and Team::releaseFrontMember() to fix**
delete the Tournament pointer	n/a	~Tournament()	deletes Tournament and Teams contained within	Segmentation fault caused by also deleting teams from main(); stopped deleting from main() and let ~Tournament() take care of it
user enters Creature type	integers 1-5	main(), mainMenu()	correct Creature is sent to Team::addMember()	correct Creature is sent to Team::addMember()
user chooses to print standings	integer 1	main(), printMenu()	standings prints and program closes	standings printed but printMenu() ran again; fixed by altering loop criteria in printMenu()
user enters more than 20 fighters per team	20	main(), intakeInt()	user is asked to input lower number	user is asked to input lower number
user enters fewer than 3 total fighters and prints top 3	**n/a**	**main(), intakeInt()**	**only 2 fighters are displayed in top 3**	**Segmentation fault trying to access absent 3rd fighter; fixed by adding a number of fighters variable to Tournament object and creating rules in printWinner() to only**

Test Case	Input Values	Driver Function	Expected Outcomes	Observed Outcomes
				display total number of fighters if less than 3
Tournament with same type of fighters on both sides (i.e. all Goblins)	n/a	doTourney()	Either team can win; final score somewhat close	Either team can win; final score somewhat close
Tournament with lop-sided fighters (i.e. Goblins and Barbarians vs Bluemen and Reptiles)	n/a	doTourney()	Stronger team always wins; final score not close	Stronger team always wins; final score not close
Tournament same composition of fighters on both sides (i.e. one of each on both teams)	n/a	doTourney()	Either team can win; final score somewhat close	Either team can win; final score somewhat close