hw02: functions, structs, vectors

Hide Assignment Information

Instructions

hw02: Functions, structs, vectors and file I/O

Focus

- Functions
- Structs
- Vectors
- I/O

Problem:

We will model a game of medieval times. Our world is filled with warriors. Naturally what warriors like to do is fight. To the death. So we happily let them.

Each warrior starts out with a name and a certain amount of strength. Each time he fights, he loses some strength. (He gets to keep his name.) If his opponent is stronger than he is, then he loses *all* of his strength, in which case he is dead, or at the very least pretty useless as a fighter. Otherwise he loses as much strength as his opponent *had*. Of course, if he and his opponent had the same strength then they are both losers.

Even losers are allowed to pick a fight. It doesn't require having any strength in order to do battle with someone else. Not that you stand much of a chance of winning anything, but perhaps it's worth getting beaten (again) just to have those 15 nanoseconds of fame.

Your program will read in a file of commands. There are three types of commands:

- `Warrior` creates a new warrior with the specified name and strength.
- `Battle` causes a battle to occur between two warriors.
- `Status` lists all warriors, alive or dead, and their strengths.

A sample input file looks like:

```
Warrior Jim 42
Warrior Lancelot 15
Warrior Arthur 15
Warrior Torvalds 20
Warrior Gates 8
```

```
Status
Battle Arthur Lancelot
Battle Jim Lancelot
Battle Torvalds Gates
Battle Gates Lancelot
Status
```

The name of the input file will be **warriors.txt. Do not ask the user for the name.**

Execute each command in order as you read it, so **it would be wrong** to, for example, save up all the Battle commands and execute them at the end.

The Status command displays how many warriors there, then displays each one with his strength. The Warrior command does not display anything. The Battle command displays one line to says who is fighting whom and a second line to report the results, as shown below.

The output (which would display **on the screen**) for this file should look like:

```
There are: 5 warriors
Warrior: Jim, strength: 42
Warrior: Lancelot, strength: 15
Warrior: Arthur, strength: 15
Warrior: Torvalds, strength: 20
Warrior: Gates, strength: 8
Arthur battles Lancelot
Mutual Annihilation: Arthur and Lancelot die at each other's
hands
Jim battles Lancelot
He's dead, Jim
Torvalds battles Gates
Torvalds defeats Gates
Gates battles Lancelot
Oh, NO! They're both dead! Yuck!
There are: 5 warriors
Warrior: Jim, strength: 42
Warrior: Lancelot, strength: 0
Warrior: Arthur, strength: 0
Warrior: Torvalds, strength: 12
Warrior: Gates, strength: 0
```

How to handle reading the lines

In some languages, e.g. Python, you tend to read in a whole line, then tokenize it, then convert portions of the input from string to numbers.

- As you have seen, this is **not necessary** in C++!
- When you need to read in an int, you can just read it into an int variable, instead of reading it as a string and then converting.

- Remember, you have seen this in lab.

But some lines have three strings, or just one string or two strings and an int. How can you handle that?

- First read in what the command is.
- After you know what the command is, only then read in the rest of the fields. If it is a status, there is no more to read in. If it is a battle, there will be two names. If it is a warrior there will be a name and a strength.
- Easy! Take advantage of whatever strengths your language offers!

Suppose we *were* worried that a line might have too many or too few tokens. Or maybe tokens of the wrong type. (Note we are **not** worrying about that in this assignment.) C++ does have a easy way to read in a whole line and then use the I/O system to again easily tokenize it without having to convert from strings to ints. If you are curious about this take a look at the class `istringstream`. But we probably have not covered that in class and are not expecting you to use it. (We can only cover so much!)

Additional notes

- For **this *application*** it will be necessary for the Warrior names to be unique. That is because we are referring to them simply by name. What should happen if you are adding a Warrior, but one already exists with that name?
    - Don't add the new Warrior,
    - display an error message,
    - and then keep reading in and executing the commands.
- What if a battle command says that Joe is battling Fred, but one or both of them have not been defined yet?
    - Don't have the battle (obviously)
    - display an error message,
    - and then keep reading in and executing the commands.
- **We do promise** that every command in the input file will have the correct *format*. So the Warrior command will be followed by a name and a strength; the Battle command will be followed by two names, and the Status command will not have any arguments.
- For the sequence of commands shown, your output should look exactly as shown. Please don't exercise you creativity *here*. I'd like the graders to be able to quickly determine whether your program in fact works, before they dive in to read your code. (You do get to output whatever you think is appropriate for errors. We didn't show any.)
- The output shown above as, "He's dead, Jim", is because a warrior named *Jim* is battling Lancelot but Lancelot, sadly, has already died. It is **not** meant as a phrase to be used whenever one of the participants is dead and the other is alive. Sure, in all such cases you would say "He's dead FILL_IN_THE_BLANK", where FILL_IN_THE_BLANK is the name of the person who is *not* dead. This is used **whenever one of the participants is dead and the other is alive**.
- For all assignments, please use the aspects of the language that we are focusing on in the assignment. For example, in this assignment, one of the topics is the use of struct. That means we **do *not* want you to define classes, methods, constructors, or to use data hiding** — that will all be in the next assignment. (Obviously, you shouldn't

worry if you don't know what I was just talking about there.) Why the restrictions? The purpose of each assignment is to have you exercise *specific* technical skills. If you want to wow us with the cool stuff you have figured out that goes beyond the current syllabus, great! Come by the office and I will be happy to look at and discuss your work. Just don't put it in your homework / lab submissions.

- Follow the same guidelines as in the past. Here are some reminders.
    - Make *good use* of functions. In this program, for example, when you determine that you are looking at a Warrior command then you should call a function to handle that command. Similarly with the other commands. Keep them short.
    - *Always* begin your file with a comment identifying who you are, what this program is and what this program is for.
    - If there are any *known* problems with your program include a comment explaining them. You *have* carefully tested your program, right?
    - The program should have a "reasonable" amount of *useful* comments.
    - The code must be well formatted.
        - *Never* turn in code with bad indentation.
        - Remember the length of a line should be limited to 80 characters
    - All structs, functions and variables should have *good* names.
        - Function names should clearly identify what the function *does*.
        - A variable name that is a single letter, such as `i`, should only be used within the scope of a loop as a loop *index*. Ok, yes, x and y might make excellent names for referring to the coordinate of a point in 2D space.
        - `temp` is almost **never** a good name. `flag` isn't much better.
    - Remember to use function prototypes so that the first function *definition* will be `main`.
- Follow the convention that structs begin with an uppercase letter, constants are all uppercase and functions and variables begin with a lowercase letter.

Turn in

Hand in a single cpp file, **hw02.cpp**, containing your program.

Questions?

Raise them on the Forums section of Brightspace

Due on Feb 9, 2026 11:59 PM

Available on Jan 28, 2026 8:00 AM. **Access restricted before availability starts.**

Attachments

warriors.txt (202 Bytes)

Download All Files

Submit Assignment