Boom!

Due Tuesday, April 8 at 8 a.m.

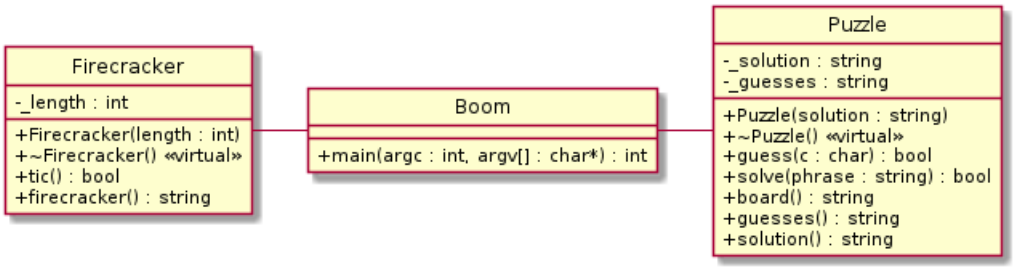
CSE 1325 - Spring 2025 - Homework #9 - Rev 3

Assignment Overview

You created your own types in Java. Why should C++ be any different! Boom! Let's OO in C++!

Full Credit

Consider the following class diagram, which implements a word guessing game in which the Firecracker burns one segment for each letter guessed and missed, and the firecracker goes “Boom!” if the Firecracker is expended before the phrase is guessed. The player may guess the solution at any time, winning if correct or immediately getting a “Boom!” if incorrect.



class Firecracker

This models a firecracker with a fuse that burns up a segment each time `tic` is called. `tic` is false if the fuse is consumed and the firecracker goes boom. **This class is declared in file firecracker.h and implemented in file firecracker.cpp.**

Firecracker

Initialize the field to the parameter *using an init list*. If `length` is less than 3, throw a `std::invalid_argument`.

~Firecracker

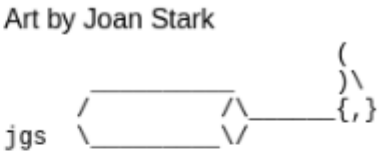
This is the destructor, declared as simply `virtual ~Firecracker();` and implemented as simply `Firecracker::~Firecracker() { }`. They keyword `virtual` is only used in the header file - it signifies that superclass destructors should also be invoked on a `delete`. (We have no superclass here, but it's a good habit to always declare this.)

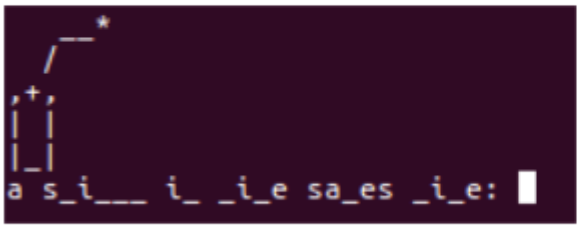
tic

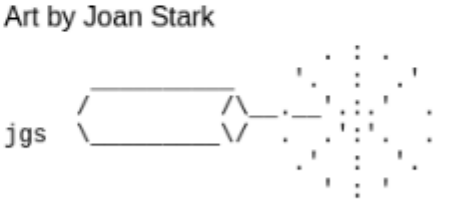
If `_length` is not already 0, decrement `_length`. Then return the Boolean `_length > 0`.

firecracker

Return a string representation of a firecracker. Whatever you like is fine, however, the number of segments remaining on the Firecracker's fuse must match `_length`. Here's a few ideas to get you started.







class Puzzle

This models the game puzzle - the phrase that the player is trying to guess before the firecracker goes boom. **This class is declared in file puzzle.h and implemented in file puzzle.cpp.**

Puzzle and ~Puzzle

Initialize `_solution` to the parameter and `_guesses` to a space *using an init list*. If `solution` is empty, throw a `std::invalid_argument`. The destructor `~Puzzle` follows the pattern of `~Firecracker`.

guess

Ensure c is lower case. If c is not alphabetic or has already been guessed (that is, c is already in `_guesses`), throw a `std::invalid_argument`. Otherwise, append c to `_guesses`. Then return true if the guess is in `_solution` (good guess!), false otherwise (you guessed... poorly).

solve

This is an attempt to solve the puzzle. Return true if the parameter matches `_solution`, false otherwise.

board

This is the playing board. Return a string with correctly-guessed letters shown and not-yet-guessed letters replaced with an underscore (_). The purple firecracker screenshot above shows the board for an in-progress game for the puzzle `a stitch in time saves nine`.

solution

This is a getter. Simply return `_solution`.

guesses

This is a getter. Simply return `_guesses`.

Boom

This is the main function, **implemented in file boom.cpp.**

- Print a banner screen and instructions if you like.
- Construct a new `Firecracker` with 8 segments and a new `Puzzle` using `argv[1]` as the secret phrase.
- Enter the main loop.
    - Show the firecracker from `Firecracker` and the letters already guessed from `Puzzle`. Ask for and accept a single char - the guess unless '0' or '!'.
    - If the letter is '0', exit. The player gives up and loses.
    - If the letter is '!', accept a string as a guess as to the solution. If correct (that is, `puzzle.solve` is true), the player wins. If incorrect, the player loses.
    - Otherwise, pass the letter to `puzzle.guess`. If the response is false (the letter is not in the puzzle), call `Firecracker.tic`. If `Firecracker.tic` returns false, the player loses, otherwise continue playing. Be sure to catch and handle any exceptions thrown by invalid guesses!
- When the loop exits, report if the player won or lost.

Running Boom!

The program you wrote (as you can tell) requires a single argument, the phrase to be guessed. This phrase almost always contains spaces. Since spaces are also used to separate arguments on the command line, you need to include apostrophes around the phrase, like this:

```
./boom 'now is the time for all good men to come to the aid of their country'
```

Hints

Finding a char in a string

You'll need to be able to determine if a `char` is in a `string` - for example, when determining if a letter has already been guessed (in method `guess`), returning from `guess` if the letter was part of the `_solution`, and in method `board` when deciding if a letter should be shown or replaced with "_".

You have two options, both of which are perfectly acceptable *for this assignment*.

- You may write a *private* static `bool contains(char c, std::string s)` method that returns true if `s` contains `c` and false otherwise, by iterating over `s` and checking each char. This is good practice, but otherwise suboptimal because `std::string`'s *know* how to check if they contain a character!
- You may use the `std::string::find` method, which returns the position of the first char found that matches, or `npos` (no position) otherwise. `s.find(c) == std::string::npos` will be true if `c` is NOT in `s`, false if `c` IS in `s`. This is the approach the suggested solution will demonstrate.

See https://cplusplus.com/reference/string/string/npos/ for documentation on `npos`.

Because `npos` is a static member of string, we can access it at the class level. In C++, this looks like `std::string::npos` - "the npos that belongs to string, which belongs to std".

Determining if a char is alpha

You'll also need to determine if a char is alpha - that is, it is in the range a-z or A-Z.

C offers the `isalpha(c)` function, which we will accept here.

Better is to use the more sophisticated C++ `std::isalpha`, which is international.

- A simple `std::isalpha(c)` usually works.
- The more proper use is `std::isalpha(static_cast<unsigned char>(character))`. You needn't use this, since we haven't discussed `static_cast` yet. But feel free if you like.
- If you'd like to make Boom! international, you can also select a locale

Makefile

Here's a Makefile that should work for all levels of this assignment. Remember that the first character of indented lines must be a TAB, not a SPACE!

```
CXXFLAGS = --std=c++20

boom: boom.o puzzle.o fuse.o
	-$(CXX) $(CXXFLAGS) boom.o puzzle.o fuse.o -o boom
	@printf "Now type ./boom 'phrase to guess' to play the game!\n\n"

fuse.o: fuse.cpp *.h
	-$(CXX) $(CXXFLAGS) -c fuse.cpp -o fuse.o

boom.o: boom.cpp *.h
	-$(CXX) $(CXXFLAGS) -c boom.cpp -o boom.o

puzzle.o: puzzle.cpp *.h
	-$(CXX) $(CXXFLAGS) -c puzzle.cpp -o puzzle.o

clean:
	rm -f *.o *.gch a.out  boom
```

Bonus

We will cover sets in Lecture 20, so (once you've completed the full credit version), let's put them to use! We'll replace the `std::string _guesses` in the `Puzzle` class with a `std::set<char> _guesses`.

Since `std::set` is sorted by default, this has the delightful side-effect of making our "Guessed" listing on the game board sorted.

Simply work your way through puzzle.h and puzzle.cpp, making the change. **Keep the public interface in puzzle.h identical**, so that no other files will need to change.

Here's a summary of the changes made in the suggested solution:

puzzle.h

- Change the `_guesses` field to type `std::set<char>`. Note that you'll need to include `<set>` to make `std::set` visible.

puzzle.cpp

- In the constructor, remove `_guesses` from the init list. Add a statement in the body to insert a space into the set so that spaces in the puzzle `board()` are visible.
- In method `guess`, determine if c has been guessed using the `count` method - if the count isn't zero, c has been guessed. Also, of course, insert c into the set instead of concatenating it into the string.
- In method `board`, use method `count` again to check if each letter in `_solution` has been guessed yet.
- In method `guesses`, convert the `set` into a `string`. C++ has sophisticated library routines for this, but let's keep it simple here: Just iterate over the set using a for-each loop and concatenate each `char` to a string. Return the string.

That's it! The other classes should require no changes, since your public interface didn't change. Did you notice how easy it is to verify that, since the *entire public interface* is separately declared in the `public` section of the class header file? :)

Extreme Bonus

It's a bit annoying to type '!' and enter the correct phrase once all of the letters have been revealed. Modify the program so that when all letters have been revealed, you are immediately declared the winner!

One thought is a boolean method for `Puzzle` named `solved` that checks each letter in `_solution` for membership in `_guesses`, returning true if they're all there and false if any are missing.

But if you have a better idea, go forth and `solve` it!