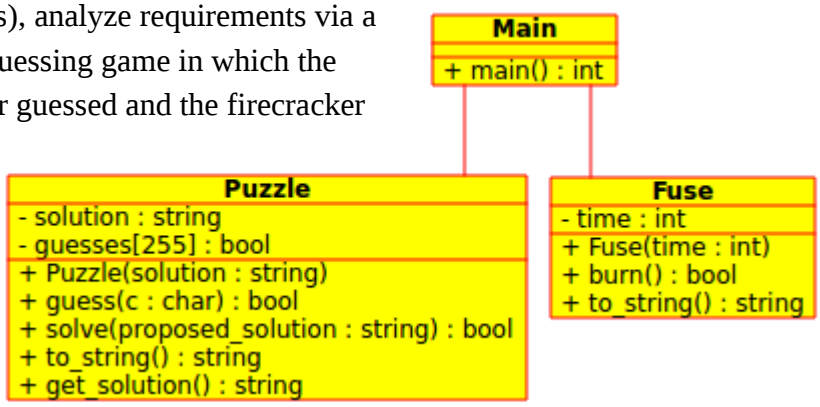
Raising Your Skill – Boom

CSE 1325 – Spring 2018 – Homework #3
Due Thursday, February 8 at 8:00 am

Assignment Overview

This assignment moves us past simple single-file programs into more interesting multi-file programs. We also write a game that, if not a viable League of Legends™ competitor, is at least reasonably enjoyable at the Extreme Bonus level. Along the way, we'll continue to utilize a Makefile, git, and Umbrello to design our class and use case diagrams, and add the ddd debugger to our toolkit.

Full Credit: Using git (3+ commits), analyze requirements via a UML Use Case diagram for a word guessing game in which the fuse burns one segment for each letter guessed and the firecracker goes "Boom!" when the fuse is expended. The player may guess the solution at any time, winning if correct or immediately getting a "Boom!" if incorrect. Implement in C++ using the multiple classes in the UML class diagram to the right. Run the program in the ddd debugger and view the contents of the guesses array during a game.



Deliver file CSE1325_03.zip to Blackboard. In **subdirectory "full_credit"**, include your full git repository, including files **main.cpp, puzzle.h, puzzle.cpp, fuse.h**, **fuse.cpp, Makefile, puzzle.xmi** (Use Case diagram with a **puzzle-usecase.png** screenshot), and screenshots **puzzle.png** (game in progress), **puzzle-ddd.png** (debugger session above), and **puzzle-git.png** (git log).

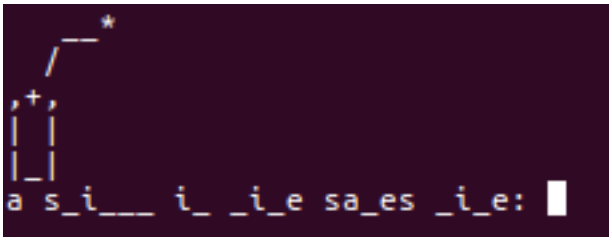
Bonus: Using git (3+ additional commits) and the above code, **update the code to not burn a fuse segment on a correct guess**. Also throw a custom exception on a repeated guess to avoid burning fuse. In CSE1325_03.zip **subdirectory "bonus"**, include **main.cpp, puzzle.h, puzzle.cpp, fuse.h, fuse.cpp, Makefile**, and **puzzle-git.png** (a screenshot of the git log).

Extreme Bonus: Using git (3+ additional commits), add a new class Words that accepts a filename as constructor parameter, and returns a random word from that file on each call to its get_word() method. Modify main() to accept a filename as a command line parameter, and set the solution randomly via Words. In CSE1325_03.zip **subdirectory "extreme_bonus"**, include all files required for Bonus above as well as **words.h, words.cpp**, and **words.txt** (a list of word puzzles[1]).

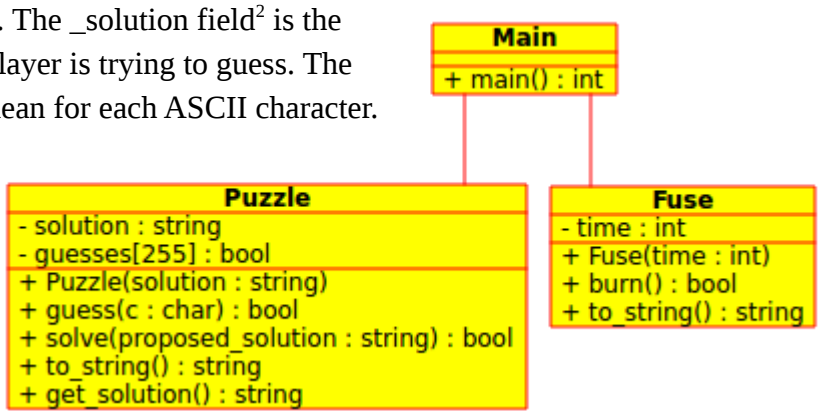
1 Ever wondered what's in the /usr/share/dict directory in Linux? Just a random question...

Full Credit Requirements

Boom is a classic word or phrase guessing game with limited tries and something catastrophic if the word is missed – a firecracker goes (you guessed it) “Boom!”. A game in progress looks something like the screen shot to the right.



Below is a simple UML class diagram of the Full Credit version of the game. Note that Main isn't actually a class – it’s simply the usual main function. This time around, implement a .h and .cpp for each class (as discussed in Lecture 5).



The Puzzle class represents the game. The _solution field[2] is the string that is the word or phrase the player is trying to guess. The _guesses vector or array holds a Boolean for each ASCII character. For example, _guesses[‘a’] is true if ‘a’ has been guessed, false if not.

The Puzzle constructor accepts the solution, which should be stored in _solution. The guess method accepts a single character (the player’s guess), and returns true if the character is valid (between ‘a’ and ‘z’, and not yet guessed) and false otherwise. It should also update _guesses. The solve method accepts a proposed solution, and returns true if correct (it matches _solution) or false if incorrect. The to_string method returns the player’s view of the puzzle – that is, characters that have been guessed are visible, those that have not been guessed are replaced with a ‘_’, and spaces are not changed. The get_solution getter method simply returns _solution.

The Fuse constructor accepts the time (i.e., number of guesses) that may elapse before the firecracker goes boom. The burn method decrements the remaining time, and returns true if any time remains or false if time has expired. The to_string method returns an ASCII art representation of the firecracker, with the number of segments of fuse representing the time (aka number of guesses) remaining. You are free to create your own ASCII art representation of the firecracker, or use the one above.

Main will implement the following algorithm:

(1) Create the variables.

---

2 For all fields, you may use or omit the leading underscore, as you please. The UML omits them.

(2) Display a welcome message, like this:

```
          ================
              B O O M !
          ================

Enter lower case letters to guess,
! to propose a solution,
0 to exit.
```

(3) Begin the main loop:

(3a) Display the firecracker and the player's view of the puzzle, with a prompt, something like this:

```
    ________*
   /
,+,
| |
|_|
_ ______ __ ____ _____ ____:
```

(3b) Accept a character from the player. If it's a 0, immediately exit. If it's an exclamation point (!), ask for a proposed solution: If correct, they win, if not, they lose. If the character is an invalid guess, print "Invalid character - try again". Otherwise, update the state of the puzzle using the player's guess, including decrementing the length of the fuse.

(4) After a proposed solution, if the player won, display something like this:

```
*** W I N N E R ***
```

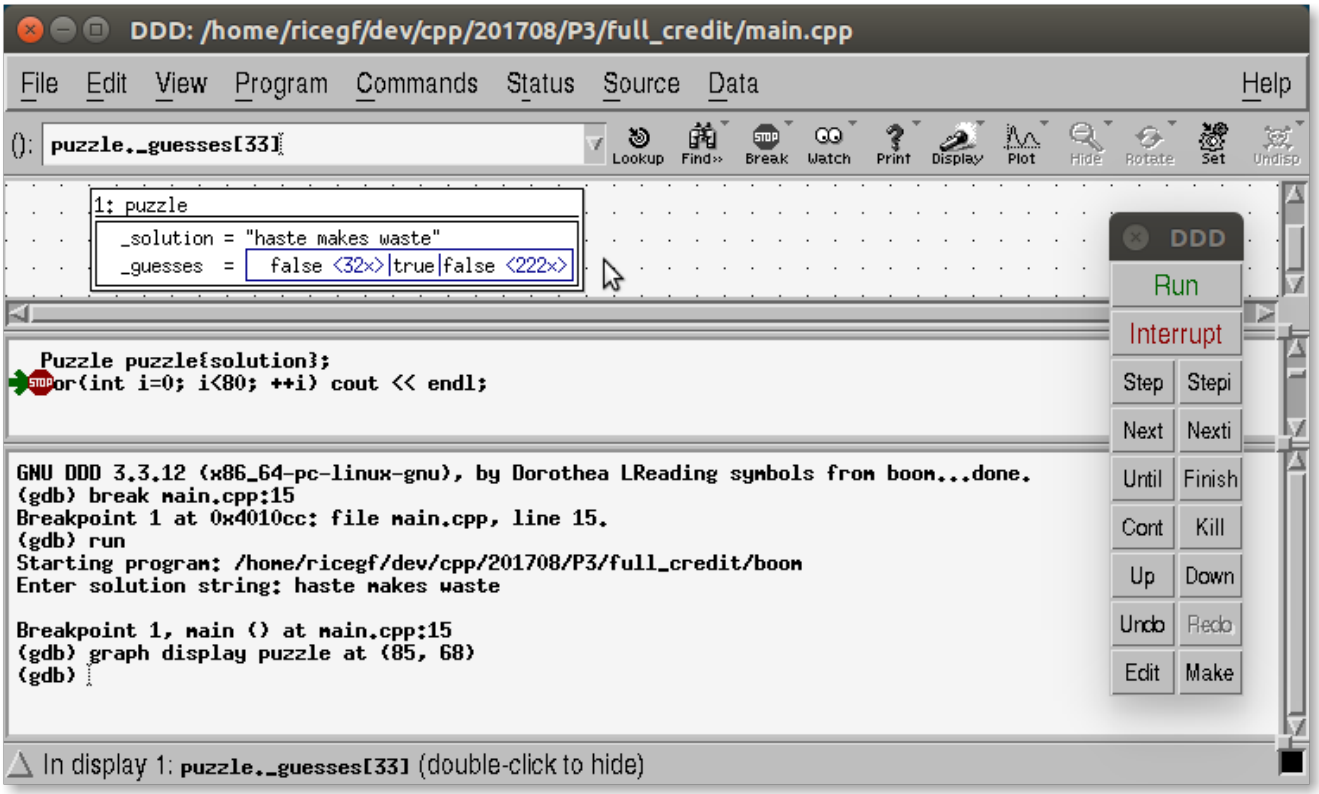
and something like this if the player lost by missing a guess or running out of fuse:

```
###### BOOM ######
The answer was: a stitch in time saves nine
```

At this level, it is acceptable to allow one player to provide the puzzle solution before the game begins (hint: output 80 newlines to hide the solution), then allow a second player to play.

The student will create a simple Use Case diagram in Umbrello or similar tool for the game with one actor (Player) and three use cases: Guess a Letter, Guess the Solution, and Exit. No additional documentation (diagrams or text) is required for each use case, and you needn't recreate the class diagram above yet.

Once the program is running, build it with the -g option. Open in the debugger and set the solution string to include your full name. Run until the Puzzle class instance has been constructed (think breakpoint, single step, etc.). Set the debugger window such that the Puzzle class instance can be viewed in the watch window (View → Data Window), exposing its private variables, take a screenshot of the entire window, and name it puzzle-ddd.png, like this:



As with previous homework, the student will deliver a ZIP archive file named **CSE1325_03.zip** with full_credit, bonus, and extreme_bonus subdirectories.

The full_credit subdirectory will contain the local git repository as usual, along with the following additional files :

- The source code files **main.cpp, puzzle.h, puzzle.cpp, fuse.h**, and **fuse.cpp,**
- A correct **Makefile**,
- A **puzzle.png** image file demonstrating the game in progress,
- A **puzzle-usecase.png** image of and **puzzle.xmi** Umbrello file containing the use case diagram[3],
- The **puzzle-ddd.png** file created above of the debugger showing the vector, and
- A **puzzle-git.png** image file demonstrating at least 3 commits to git during development (the actual number of commits and commit messages may vary).

3 It's fine if your puzzle.xmi already includes the class diagram from the Extreme Bonus. If you use an on-line tool, include a file puzzle.url with the link to your public model instead.

Bonus Requirements

If you played a similar game in school, you probably didn't use one of your fuse links if the character you guessed was part of the word.

At this level, Puzzle::guess will be modified such that returning true indicates the guessed character was indeed in the solution, while returning false indicates that it was not (and thus the fuse length on the firecracker should decrement). If an invalid guess is submitted, **throw a custom exception**[4].

As part of your algorithm in main, catch the exception and respond exactly as in the full credit version – with the error message "Invalid character - try again".

For maximum credit, the bonus subdirectory will contain the following source code files corresponding to the class diagram:

- The source code files **main.cpp, puzzle.h, puzzle.cpp, fuse.h,** and **fuse.cpp,**
- A correct **Makefile**,
- A **puzzle-git.png** image file demonstrating at least 3 commits to git during development.

4 Check Lecture 4 for an example of defining and throwing a custom exception.

Extreme Bonus Requirements

A game with only one puzzle doesn't have much replay value, and you don't always have a friend to enter one for you. It's better for the game to pick a word or phrase at random.

For the extreme bonus, **add a new class, Words**, to add the spice of variety to the game. The constructor for Words will accept a filename, and will open that file and load all of the words (one per line) into the private vector of strings named _words. For this homework, bash can't save us – we need cin to accept keyboard input to play the game, so **we'll need to actually open the text file as a new stream** in order to load it into the _words vector. Stack Overflow or CPlusPlus.com is your best friend here. The Words class will contain a single method, get_word, that returns one of these words chosen at random from the file specified as the constructor's parameter as a string. Consider the case where the filename doesn't exist or otherwise can't be fully read, and respond appropriately per Lecture 4.

Now modify main to let the user specify the filename – either accept as an optional parameter the filename of one of the above list of words or phrases (including possibly spaces), exactly as you did in C long ago (think argv and argc), or just ask for the filename. Use the Words class to load it into memory and to select one of the words or phrases at random to use as the solution parameter to the Puzzle constructor.

While you're here, also **improve the game by checking whether all letters have been guessed correctly**; if so, then don't make the player enter their solution, just declare them the winner! This may require an extra method in puzzle.

For maximum credit, the extreme_bonus subdirectory will contain the following:

- The source code files **main.cpp, puzzle.h, puzzle.cpp, fuse.h, fuse.cpp, words.h**, and **words.cpp,**
- A correct **Makefile**,
- A **puzzle-class.png** image of and **puzzle.xmi** Umbrello file (or equivalent) containing the updated class diagram in addition to the earlier use case diagram, and
- The **words.txt** file that you used to test the extreme bonus version of your program.