# Cube

- **Due: Tuesday, March 21st, 11:59 p.m.**
- Files: cube.c, cube.h, wizard.c, wizard.h (download from Canvas)
- It is highly recommended to team up with another student for this assignment.
- Maximum number of students within a group: 2
- It is prohibited to look at the source code of another team, but you are encouraged to discuss problems using a whiteboard.
- This project may be significantly more challenging than the first assignments. Please start early!

## A. Description

In this assignment you will have to develop a multi-threaded game simulator, called **cube** (inspired by the "Cube" movie. Original project specifications courtesy of Prof. Giovanni Vigna). The cube is a structure of *n* by *n* rooms. Each room has four doors, each located on one of the walls (North, South, East, West). The rooms are bent in space so that they are structured in a toroidal fashion (that is, the rooms are "wrapped around").

The doors of a room automatically lock when two people are in the room. The doors are unlocked otherwise.

The cube is inhabited by wizards, which can move from room to room in no particular order. This means that a particular wizard may be sometimes faster and perform a move before another wizard can make a move, i.e. wizards do not take turns when moving.

A wizard can cast a spell on another wizard. If the spell is successful the victim wizard freezes. A wizard can also cast a "wakeup" spell on a frozen wizard, which will bring the wizard back to life.

The wizards are organized in two competing teams. A team wins when all of the wizards of the opposing team are frozen. Once this happens, you must terminate all threads. Check what threads are still running by running:

```
$ps -eLf | grep <your-username>
```

on the same machine, but different terminal, before entering 'exit'.

e.g. During a game run with 10 wizards, you will have 11 threads (10 wizards + a thread for the cube):

```
[yourid@c4lab15 ~]$ ps -eLf | grep yourid
root      1729  2284  1729  0    1 05:12 ?        00:00:00 sshd: yourid [priv]
yourid    1731  1729  1731  0    1 05:12 ?        00:00:00 sshd: yourid@pts/2
yourid    1732  1731  1732  0    1 05:12 pts/2    00:00:00 -tcsh
root      1810  2284  1810  0    1 05:24 ?        00:00:00 sshd: yourid [priv]
yourid    1812  1810  1812  0    1 05:24 ?        00:00:00 sshd: yourid@pts/3
yourid    1813  1812  1813  0    1 05:24 pts/3    00:00:00 -tcsh
yourid    1943  1732  1943  0   11 05:28 pts/2    00:00:00 ./cube
yourid    1943  1732  1950 24   11 05:29 pts/2    00:00:01 ./cube
yourid    1943  1732  1951 33   11 05:29 pts/2    00:00:02 ./cube
yourid    1943  1732  1952 20   11 05:29 pts/2    00:00:01 ./cube
yourid    1943  1732  1953 29   11 05:29 pts/2    00:00:02 ./cube
yourid    1943  1732  1955 23   11 05:29 pts/2    00:00:01 ./cube
yourid    1943  1732  1956 23   11 05:29 pts/2    00:00:01 ./cube
yourid    1943  1732  1958 20   11 05:29 pts/2    00:00:01 ./cube
yourid    1943  1732  1959 22   11 05:29 pts/2    00:00:01 ./cube
yourid    1943  1732  1960 35   11 05:29 pts/2    00:00:02 ./cube
yourid    1943  1732  1961 23   11 05:29 pts/2    00:00:01 ./cube
yourid    1964  1813  1964  0    1 05:29 pts/3    00:00:00 ps -eLf
yourid    1965  1813  1965  0    1 05:29 pts/3    00:00:00 grep yourid
```

After the winner was declared, but before we exit (1 thread):

```
[yourid@c4labpc15 ~]$ ps -eLf | grep yourid
root      1729  2284  1729  0    1 05:12 ?        00:00:00 sshd: yourid[priv]
yourid    1731  1729  1731  0    1 05:12 ?        00:00:00 sshd: yourid@pts/2
yourid    1732  1731  1732  0    1 05:12 pts/2    00:00:00 -tcsh
root      1810  2284  1810  0    1 05:24 ?        00:00:00 sshd: yourid[priv]
yourid    1812  1810  1812  0    1 05:24 ?        00:00:00 sshd: yourid@pts/3
yourid    1813  1812  1813  0    1 05:24 pts/3    00:00:00 -tcsh
yourid    1943  1732  1943 85    1 05:28 pts/2    00:02:43 ./cube
yourid    1972  1813  1972  0    1 05:32 pts/3    00:00:00 ps -eLf
yourid    1973  1813  1973  0    1 05:32 pts/3    00:00:00 grep yourid
```

When two wizards are in a room at the same time many things may happen:

- If the two wizards are from opposite teams, they will engage in a magic fight, whose outcome is determined by a random function. Whoever loses the fight will freeze, possibly forever.
- If the two wizards are from the same team and they are both active, they will simply waste some (random) amount of time bragging about their adventures in the Cube. If one of the wizards is frozen, then his friend will try to unfreeze him using a wake up spell, whose outcome is determined by a random value.

The skeleton of the application is provided to you. You have to develop the parts that are flagged as missing and, of course, deal with the synchronization issues.

In addition, you will have to devote a thread to interact with the user. The thread will provide a user with a prompt ("**cube**> ", to be precise), and the user can enter commands using that prompt. If the user types ‘`s`’, it will single-step through the game, only printing one move and then pausing for further input. If the user types ‘`c`’, it will continue to the end of the game only printing the resulting cube. In particular, once the cube is "initialized" with the players, the execution can be started using either the ‘`s`’ (single-step) or ‘`c`’ (continuous) command. The user can then interact with an ongoing game through the prompt. More precisely by entering ‘`show`’, the user can request to print an ASCII representation of the game.

For example, by typing `show` the thread will print something like this:

```
+--+--+--+--+--+
|aB|BB|  |  |  |
+--+--+--+--+--+
|  |A |  |  |  |
+--+--+--+--+--+
|  |b |  |B |  |
+--+--+--+--+--+
|  |  |A |  |a |
+--+--+--+--+--+
|  |Ba|  |  |AA|
+--+--+--+--+--+
```

In this figure, the 5x5 cube is populated with wizards from team `A` and team `B`. A wizard from a team is shown with the team's name uppercase if active or with the team's name lowercase if frozen.

The user can exit at any moment typing the command ‘`exit`’.

The game is invoked by calling `cube` with the following command line parameters:

`-size`

The size of the cube

`-seed`

The seed used for random number generation

`-teamA`

The number of players for team A

`-teamB`

The number of players for team B

**B. Output**

It is crucial for credit (and for debugging purposes) to **strictly** follow the following output convention:

*Single Step Example:*

Sample program output (positions are in (column, row) order:              Comments (for your reference):

```
$ ./cube -size 2 -seed 2 -teamA 1 -teamB 2
cube>show
+--+--+
|  |A |
+--+--+
|  |BB|
+--+--+
cube>s
Wizard A0 in room (1,0) wants to go to room (0,0)      Simple Move
Wizard A0 in room (1,0) moves to room (0,0)
Wizard A0 in room (0,0) finds nobody around
cube>s
Wizard B1 in room (1,1) wants to go to room (1,0)      Simple Move
Wizard B1 in room (1,1) moves to room (1,0)
Wizard B1 in room (1,0) finds nobody around
cube>s
Wizard B1 in room (1,0) wants to go to room (1,1)      Simple Move
Wizard B1 in room (1,0) moves to room (1,1)            (due to race condition, B1
Wizard B1 in room (1,1) finds nobody around             moves again)
cube>s
Wizard B0 in room (1,1) wants to go to room (0,1)      Simple Move
Wizard B0 in room (1,1) moves to room (0,1)
Wizard B0 in room (0,1) finds nobody around
cube>s
Wizard A0 in room (0,0) wants to go to room (0,1)      Battle Enemy
Wizard A0 in room (0,0) moves to room (0,1)
Wizard A0 in room (0,0) finds enemy B0
Wizard A0 in room (0,1) freezes enemy B0
cube>show
+--+--+
|  |  |
+--+--+
|Ab|B |
+--+--+
cube>s
Wizard B1 in room (1,1) wants to go to room (0,1)     Failed Move
Request denied, room locked!
cube>s
Wizard A0 in room (0,1) wants to go to room (0,0)     Simple Move
Wizard A0 in room (0,1) moves to room (0,0)
cube>s
Wizard B1 in room (1,1) wants to go to room (0,1)     Help Friend
Wizard B1 in room (1,1) moves to room (0,1)
Wizard B1 in room (0,1) unfreezes friend B1
cube>s
Wizard B0 in room (0,1) wants to go to room (0,0)     Battle Enemy and win game
Wizard B0 in room (0,1) moves to room (0,0)
Wizard B0 in room (0,0) freezes enemy A0
Team B won the game!
cube>show
+--+--+
|aB|  |
+--+--+
|B |  |
+--+--+
```

*Continue to End Example:*

Sample program output: Comments (for your reference):

```
$ ./cube –size 2 –seed 2 –teamA 1 –teamB 2
cube>show
+--+--+
|  |A |
+--+--+
|  |BB|
+--+--+
cube>c
Wizard A0 in room (1,0) wants to go to room (0,0)     Simple Move
Wizard A0 in room (1,0) moves to room (0,0)
Wizard A0 in room (0,0) finds nobody around
Wizard B1 in room (1,1) wants to go to room (1,0)     Simple Move
Wizard B1 in room (1,1) moves to room (1,0)
Wizard B1 in room (1,0) finds nobody around
Wizard B1 in room (1,0) wants to go to room (1,1)     Simple Move
Wizard B1 in room (1,0) moves to room (1,1)           (due to race condition, B1
Wizard B1 in room (1,1) finds nobody around            moves again)
Wizard B0 in room (1,1) wants to go to room (0,1)     Simple Move
Wizard B0 in room (1,1) moves to room (0,1)
Wizard B0 in room (0,1) finds nobody around
Wizard A0 in room (0,0) wants to go to room (0,1)     Battle Enemy
Wizard A0 in room (0,0) moves to room (0,1)
Wizard A0 in room (0,0) finds enemy B0
Wizard A0 in room (0,1) freezes enemy B0
Wizard B1 in room (1,1) wants to go to room (0,1)     Failed Move
```

```
Request denied, room locked!
Wizard A0 in room (0,1) wants to go to room (0,0)      Simple Move
Wizard A0 in room (0,1) moves to room (0,0)
Wizard B1 in room (1,1) wants to go to room (0,1)      Help Friend
Wizard B1 in room (1,1) moves to room (0,1)
Wizard B1 in room (0,1) unfreezes friend B1
Wizard B0 in room (0,1) wants to go to room (0,0)      Battle Enemy and win game
Wizard B0 in room (0,1) moves to room (0,0)
Wizard B0 in room (0,0) freezes enemy A0
Team B won the game!            <--- At this point all wizard threads should be terminated!!!!
cube>show
+--+--+
|aB|  |
+--+--+
|B |  |
+--+--+
cube>exit
```

Although not shown in the single step example above, '`show`' should print the current state of the cube at the given time.

## C. Provided Infrastructure

You are provided with part of the code, and should download these files from Canvas. You will have to complete the existing code to implement the multithreaded part of the game. Comments such as

```
/* Fill in */
```

are placed in the code to indicate that some relevant code has been removed from the source file. This is the code you should provide. Note that, since there are infinite slightly different possible solutions it may be the case that your application will work without having to complete all the missing parts. Your solution may also add code to any part of the source.

The provided code base is composed of the following files:

- cube.h: definitions for the cube, rooms, and wizards. You can only add the needed thread-related variables to these definitions.
- wizard.h: prototype of wizard function. You should not modify this file.
- cube.c: implementation of most of the game.
- wizard.c: implementation of the wizard function.

When you compile cube, you might want to include `readline, history` libraries with `-l`, like this:

```
gcc -g cube.c wizard.c -lreadline -lhistory -lncurses -lpthread -o cube
```

Please monitor the class mailing list closely for any variations and modifications to these files.

Your program will be evaluated by both manual inspection and automated scripts. It is therefore VERY important that it compiles without problems and that it is resilient to unexpected output.

## D. Submitting your assignment

Submit the files individually on Canvas (5 total: cube.c, cube.h, wizard.c, wizard.h, and makefile).

**F. Mistakes to avoid from the past**

- One of the trickiest cases used in the past was the following:
  ```
  $ ./cube -size 2 -seed 2 -teamA 4 -teamB 3
  ```
  This should not cause a deadlock! We will never use a test case where:
  $$\Sigma(\text{WizardsA}) + \Sigma(\text{WizardsB}) \geq n \text{ (where } n = \text{size)}$$
- Backup your work frequently! Do this saving files every day with a different file name (i.e. backup-osproject3-Mar2)
- Be thorough and ensure that your wizards move according to the rules of the game by single stepping through your code!
- Ensure that you notice exit conditions (i.e. all wizards from enemy camp are frozen) and that you terminate your threads BEFORE you actually exit your program.