Programming Assignment 4

Due Wednedsay, October 14 **in class**

NAME

`elevators` – multithreaded elevator simulation

USAGE

```
elevators [-e nelevators] [-s speed] [-g]
```

DESCRIPTION

This program simulates a building with elevators and people.

`-e nelevators` — number of elevators in the simulation

`-s speed` — time (in seconds) between simulation ticks

`-g` — format output for graphical display

The building has 11 floors numbered from 0 to 10. People enter the building on floor 1 at various times. They take elevators to various floors, spend time working on those floors, and eventually exit the building on floor 1. The people and their behavior are described by a series of single line text records fed to the simulation on stdin. Each person record is a line:

```
name starttime floor1 worktime1 ... floorN worktimeN
```

The person will enter the first floor at `starttime`, proceed to `floor1`, work for `worktime1`, then proceed to `floor2` and so on until `floorN`, where they work for `worktimeN` and then take an elevator to the first floor to exit.

Each elevator begins at floor 1. Elevators can hold an unlimited number of people. It takes one tick for an elevator to move up or down one floor. It takes one tick for the elevator's door to open, and one tick for the elevator's door to close.

A person can only enter an elevator when the elevator's door is open. An elevator can only move between floors when its door is closed. It takes no time for a person to enter an elevator.

The code for the building is written for you. The building will create the people and elevators, handle the I/O, and calculate statistics. Your job is to write the elevator controller, and manage the interactions between people and elevators.

Unlike a real building, people announce their destination when requesting an elevator. Unlike a real building, elevators allow specific people to board – the people have no option to board an elevator that happens to be on their floor.

ELEVATORS DISTRIBUTION

The elevators distribution contains the source files for the elevator simulation and a number of helper and utility files. It is on github at:

`https://github.com/turtlegraphics/elevators`

Your team's repo (TeamXX-elevators) is a copy of the base distro.

SOURCE FILES

`makefile`
: Type `make elevators` to compile the appropriate files. You should modify this file if you rename your code or add new modules.

`elevators.C`
: This is a skeleton solution to the problem. All the routines are defined, but do nothing. It will compile, but it is not a correct solution, because `take_elevator()` will return without the person taking an elevator.

`elevators.h`
: This header file contains declarations for the `Elevator` class, which you must implement, and the `take_elevator()` function, which you must implement. You need to implement the member functions which appear in this file (and correspond to the stubs in `elevators.C`). You should also add members to the `Elevator` class as needed for your algorithm.

`building.h` *Do not alter this file.*
: This header file is the interface between your elevator code and the building. It contains function prototypes and type definitions. The classes `ElevatorMachinery` and `Person` are defined here.

`building.C` *Do not alter this file.*
: This is the code for the building and I/O. The function `main()` is here. It does the following:

  - Create an `Elevator` object for each elevator. At this point, the constructor is called for each elevator object, giving you the opportunity to initialize per-elevator data before any threads are created.
  - Create a `Person` object for each person.
  - Create a thread for each elevator. The thread runs the function `Elevator::run()`, which you must define.
  - Create a thread for each person. The thread runs the function `Person::run()`, which is defined in `building.C`. The `Person::run()` repeatedly calls `take_elevator()` to take an elevator to a work floor. After the person finishes work, they leave the building and the thread exits. You need to write the `take_elevator()` function, which should not return until the person has gotten on an elevator and gotten off at the right floor.

- Wait for all people to exit the building, and print statistics.

Your elevators call functions in the `ElevatorMachinery` class to operate:

- `open_door()` – Call to open your elevator door. This sleeps for one tick and prints a message.
- `close_door()` – Close your door. Takes one tick.
- `move_to_floor()` – Move to a different floor. This sleeps one tick between each floor, printing messages as it moves.
- `move_up()` – Move up one floor. Takes one tick.
- `move_down()` – Move down one floor. Takes one tick.

All of these routines will print a warning message if you attempt something physically impossible, such as moving with an open door.

You can have elevators and people display messages using the `message()` functions. Don't print debugging info to stdout (e.g. with cout). Either use cerr or use the `message()` functions.

UTILITY FILES

Besides the elevator simulation source files described above, there are a few other things in the distribution.

`egraphics.py` A Python script that displays the simulation graphically.

Run `elevators` with the `-g` option and pipe the output to `egraphics.py`. Graphics requires the `cs1graphics` Python package to run. This is installed on turing, and also available on the web.

`.eld` **files** Sample person files for testing.

You might run the program: `elevators < oneguy.eld` to direct the `oneguy.eld` file to elevators standard input.

`people.C` A utility that will generate random person records.

Usage is `people [-p people] [-t trips] [-d delaymax]`

For example, `people -p3 -t8 -d10` will generate 3 people, each of which takes 8 trips and works for up to 10 ticks between trips. You can pipe the output of `people` to the input of `elevators`.

Some samples ways to run elevators using these utilities:

- `elevators -g < oneguy.eld | python egraphics.py` (one person, graphics)
- `people -p50 | elevators -n5` (50 people, five elevators)
- `elevators -n3 -s0.5 -g < tenpeople.eld | python egraphics.py` (10 people, 3 elevators, faster, with graphics)

You can also simply run `elevators` and then type a simple person record (such as 'Ann 0 5 2') followed by ctrl-D for end-of-file.

HINTS

See `/export/mathcs/home/public/clair/bin/elevators` for a working version. Get started by playing around with the various input and output options.

The source and header files have comments - read them. Read over the code in `elevator.h`, `elevator.C`, `building.h` and even `building.C` to get a sense of how the program works.

Play with the `Elevator::run()` function. Have the elevator move up and down, open and close its doors. Try an illegal maneuver, like moving with open doors.

Write a simpler version first, in which each elevator can only carry one person at a time. Here's some steps towards writing this simpler version:

1. Write a class to handle a queue of `Person` objects. You'll need to use a mutex to protect the queue operations. Have `take_elevator` put people on the queue. You can test your queue by having the elevators spin (loop), watching the queue, and then do something when a person enters the queue (like open and close their doors).

2. Add one condition variable corresponding to a non-empty queue. Change your elevators so they wait on the condition variable instead of spinning. Have the elevator dequeue a person, move to their floor, and open its doors. Get `display_passengers` working with your one "passenger".

3. Add a semaphore (or a mutex/condition variable pair) for each rider. The `take_elevator` function should not return until the elevator signals the person to get off. Elevators need to dequeue a person, move to their floor, open doors, and then signal the person to "get off".

USEFUL MAN PAGES

`pthread_mutex_init(3)`
`pthread_mutex_lock(3)`
`pthread_mutex_unlock(3)`
`pthread_cond_init(3)`
`pthread_cond_wait(3)`
`pthread_cond_signal(3)`
`pthread_cond_broadcast(3)`
`sem_init(3)`
`sem_post(3)`
`sem_wait(3)`