Coding Assignment 4

You are creating a multi threaded Trick Or Treating program.

Step 1

You will need to create 3 input files for your program. I have attached samples of all 3 files to the assignment. You should create your own files for testing. A different set of files will be for testing during the grading process.

A file of trick or treater's names - I suggest you use short names (10 characters per name) to make the display work well. Sample file is named TOT.txt.

A file of houses – this is just a list of names that will be displayed as the houses visited by the trick or treaters. Sample file is named HOUSES.txt. Names should be under 10 characters and the file should only contain at most 6 names; otherwise, the screen display will wrap.

A file of candy where each line is a candy name followed by a pipe symbol (|) and then a ranking for the candy. There can be as many candies listed in the file as you want – just make sure you have included all numbers from 1 to the number of lines in your file. Sample file is named CANDY.txt.

Remember that EOL conversions between Windows/Mac are different from UNIX and you may need to change them in your file. Notepad++ has the ability to change the EOL (Edit->EOL Conversion->Unix(LF). If you are not using Notepad++, then you can run your file through this conversion statement directly in your VM.

```
cat file.txt | tr '\r' '\n' | tr -s '\n' > file.translated.txt
```

This takes a file named `file.txt` and translates the EOL characters to LF and creates a new file called `file.translated.txt`.

Step 2

You will need 3 source files and 2 header files in your makefile. Your makefile will need to compile `Code4_xxxxxxxxxx.cpp` (referred to as just `Code4.cpp` after this point), the `TrickOrTreater.cpp` and the `House.cpp` and link them into one executable named `Code4.e`. `TrickOrTreater` will have a header file (`TrickOrTreater.h`) and `House` will have a header file (`House.h`).

You will submit 6 files in a zip file named `Code4_xxxxxxxxxx.zip`. **As always, if your `makefile` does not compile your code or your code compiles with warnings/errors, then you will be assigned a grade of 0 regardless of the rest of your work. The GTA's will not alter your `makefile`/code to make the compile work.**

```
Code4_xxxxxxxxxx.cpp

TrickOrTreater.cpp

TrickOrTreater.h

House.cpp

House.h

makefile
```

You will not submit the 3 data input files – the GTA's will grade with files that I will provide them. Do not make any assumptions about the contents of the files other than the restrictions listed above.

Step 3

Copy your function to get command line arguments from the previous assignment into this new assignment. If the function does not detect 4 command line arguments, then it should throw an invalid argument exception. `main()` should catch that exception and exit the program if any of the command line arguments are missing. The function will read 3 file names from the command line (file of Trick or Treater names, house names and candy rankings). Function should store `argv[1]`, `argv[2]` and `argv[3]` in appropriate file names that are passed back to `main()`.

Step 4

Create a map to hold the candy rankings. The ranking from the file should be the map's key and the candy's name should be the value of the map. Open the candy file and insert the information from the file into the Candy Ranking Map. You are required to use the C function `strtok()` to parse the lines from the candy file. Suggestion : temporarily add a range based for loop to print out the Candy Ranking Map to ensure that this code is functioning properly.

Step 5

Using the provided `House.h`, create `House.cpp`. A `House` object is constructed with a name and a Candy Ranking Map. A House has a member function called `ringDoorbell()` that does the following

Locks the `door` mutex

Adds an * to the passed in `ostringstream`

Sleeps the thread for 3 seconds

Randomly picks a number between 1 and the number of candies in the Candy Ranking Map

Unlocks the `door` mutex

Returns the name of the candy associated with the random number in the Candy Ranking Map.

Step 6

In `main()`, create a vector of pointers to `House`. While reading through the file of house names, use `new` to instantiate `House` objects. Each `House` object is constructed with its name and a copy of the Candy Ranking Map. This is also a good time to form the house names heading shown in the program's output. I took 11 – the length of the house's name and put that number of spaces between each name on the output screen to space them evenly. Suggestion : temporarily add a range based for loop to print out the vector of pointers to `House` objects to ensure that this code is functioning properly.

Step 7

Using the provided `TrickOrTreater.h`, create `TrickOrTreater.cpp`. A `TrickOrTreater` object is constructed with a name and a list of `House` pointers. `TrickOrTreater.cpp` contains several member functions

`getName()` – returns the name of the trick or treater

`startThread()` - see Step 9 for more information

`joinThread()` – see Step 10 for more information

`GoTrickOrTreating()` – see Step 7A for more information

`Walk()` – see Step 7B for more information getPath() – see Step 7C for more information

Step 7A

Member function GoTrickOrTreating() is the function called when the thread is started. It controls everything else the trick or treater does.

This function moves the trick or treater between houses, rings the doorbell at each house and increments the chosen candy in the trick or treater's candy bucket.

Inside a range based for loop,

a speed is determined by picking a random number between 1 and 500.

Walk() is called and is passed speed

ringDoorbell() is called and is passed path. The return value of ringDoorbell() is the name of the candy randomly chosen and that name is used to key on that candy in Bucket and increment it.

After completing the loop over all TrickOrTreater's, increment ImDone to show that this thread has visited all houses and is done trick or treating.

Step 7B

Member function Walk() uses a for loop to stream single dots '.' into ostringstream path one at a time. Between outputting each dot, the thread sleeps for the speed value passed into Walk() from GoTrickOrTreating(). See Step 7A for more information on how speed is calculated in GoTrickOrTreating().

Step 7C

Data member path is an ostringstream; therefore, member function getPath() uses path.str() to return a string containing the trick or treater's current path.

Step 8

In main(), create a vector of TrickOrTreaters. While reading through the file of trick or treater names, create a temporary trick or treater object and use push_back to add it to the vector. Each object is instantiated with a name and the list of houses. Suggestion : temporarily add a range based for loop to print out the vector of TrickOrTreaters objects to ensure that this code is functioning properly.

Step 9

In main(), use a range based for loop to call startThread() for all objects in the vector of TrickOrTreaters.

Each TrickOrTreater object contains a pointer to a thread called TOTThreadPtr. Function startThread() will instantiate a thread using new and store the pointer returned by new in TOTThreadPtr.

Step 10

In main(), use a range based for loop to call TrickOrTreater's member function joinThread(). Use the this pointer to join each object's thread with main().

Step 11

In main(), while everyone is not done with trick or treating, print

enough newlines to clear/scroll the screen the house list heading

use a range based for loop to print out the current version of each of the trick or treater's `path` and `name` (see Step 7C for more information on `path`)

sleep the `main()` thread for 5 milliseconds

Step 12

Use a range based for loop over the `TrickOrTreater`'s vector to print out the contents of each object's candy bucket. Overload << such that you can use just `cout <<` iterator and print the bucket's contents. Copy your overloaded << friend from the previous assignment and modify to print out the contents of `Bucket` which is a `map`.

Learning Outcomes that will test on Exam 2

1. compiling/linking multi module programs using a makefile
2. reading command line arguments
3. throwing/catching exceptions
4. opening files and reading their contents into various C++ containers
5. using C functions in a C++ program
6. mutex
7. map
8. vector of object pointers
9. new
10. this
11. threads
12. static variables in objects
13. operator overloading
14. friend functions