Programming Assignment IV

Due on Friday, April 1, 2016 @ 11pm

100 points

Santa Claus

Santa Claus sleeps in his shop up at the North pole, and can only be wakened up by either all nine reindeers being back from their year long vacation on the beaches of some tropical island in the South Pacific, or by some elves who are having some difficulties making the toys.

One elf's problem is never serious enough to wake up Santa (otherwise, he may ***never*** get any sleep). So, the elves visit Santa in a group of three. When three elves are having their problems solved, any other elves wishing to visit Santa must wait for those elves to return. If Santa wakes up to find three elves waiting at his shop's door, along with the last reindeer having comeback from the tropics, Santa has decided that the elves can wait until after Christmas, because it is more important to get his sleigh ready as soon as possible. It is assumed that the reindeers don't want to leave the tropics, and therefore they stay there until the last possible moment. They might not even come back, but since Santa is footing the bill for their year in paradise. This could also explain the quickness in their delivering of presents, since the reindeers cannot wait to get back to where it is warm. The penalty for the last reindeer to arrive is that it must get Santa while the others wait in a warming hut before being harnessed to the sleigh.

Here are some specifics. For each elf, he waits for two more elves in order to wake up Santa. A group of three elves submit their problems to Santa and wait until their problems are answered. Then, go back to work until a new problem comes up. While Santa is solving a problem, no other elf groups can interrupt him. More precisely, all elf groups must wait until the current group finishes working with Santa, and Santa can only work with one elf group at a time.

For each reindeer, it tans on the beaches for some time and returns to paradise. If all reindeers (say 9) have returned, the last (*i.e.*, the 9th) one wakes up Santa and all nine wait to be attached to the sleigh. Then, fly off to deliver the toys and head back to the Pacific island.

Santa takes some rests until **(1)** all reindeers have come back or **(2)** there are three elves posting questions. If it is the first case, Santa calls up all reindeer except the one who waked Santa up, setup sleigh for each reindeer. Then, the team flies off to deliver toys for a while and returns. Finally, Santa releases all reindeers. If it is the second case, Santa lets all three elves in and solves the (three) problems that the elves bring in. After solving the problem, Santa goes in bed again!

Well, Santa may want to take a long vacation as everybody does. So, Santa has made a decision that after delivering toys for some number of times (say 10), he will lay off all elves and reindeers and call for a quit. That is why you have not received toys from Santa for many years!

Each elves is a thread with the following pattern:

```
while (1) {
     Delay();                    // make toys
     AskQuestion(...);           // encounter a problem
     Delay();                    // problem solved, take a rest
     }
```

`AskQuestion()` is a function. When an elf has a question, he calls `AskQuestion()`, which blocks the calling elf until a gang of three is possible. When the control returns, the question is answered.

The following shows a reindeer thread:

```
     while (1) {
          Delay();                 // tan on the beaches
          ReindeerBack(...);       // report back to Santa
          WaitOthers(...);         // wait for others
                                   // Don't forget: the last wakes up Santa
          WaitSleigh(...)          // wait for attaching sleigh
          FlyOff(...);             // fly off to deliver toys
                                   // Santa will let go all reindeers
          Delay();                 // prepare for vacation
     }
```

`ReindeerBack()`, `WaitOthers()`, `WaitSleigh()` and `FlyOff()` are all functions. Reindeers come back one by one. When it comes back, it calls `ReindeerBack()` to register this event. Then, it calls `WaitOthers()` to wait for the other reindeers. Keep in mind that the last reindeer must wake up the Santa. After a reindeer returns from the call to `WaitOthers()`, all reindeers will act as a group. Next, all reindeers call `WaitSleigh()`, waiting to be attached to a sleigh. Again, all reindeers act as a single group. Once this step completes, all reindeers fly off as a single group to deliver toys.

The Santa thread looks like the following:

```
     while (not retire) {
          Sleep(...);              // take a nap
                                   // wakened up by elves or the last reindeer
          what is the reason?      // note that toy delivering is more important
          if (all reindeers are back) { // delivery has a higher priority
               // gather all reindeers
               // put on sleigh
               // and fly off
               Delay();            // Santa delivers toys
               // release all reindeer for vacation
          }
          if (elves have a question) { // there many be a # of groups waiting
               // let elves in
               Delay();            // solve their problem
               // solve the problem and release elves
          }
     }
```

Santa sleeps first with function `Sleep()`, which means he is blocked until three elves ask questions, or the last reindeer wakes him up. Santa always takes care the reindeers first. If Santa is wakened up by the last reindeer, he releases all waiting reindeers. Santa waits all reindeers are there. Then, he attaches the sleigh (this is the reason that all reindeers must act as a single group). After the sleigh is attached, Santa and reindeers fly off. On the other hand, if Santa is wakened up by elves, he takes some time to solve the problem, and releases all three elves.

Your implementation must correctly handle the following ***important situations***:

- Only exactly three elves can form a group and ask questions.
- Before the problem is solved, no other elves can leave. While three elves are waiting for the answers, no other elves can cut in.

- Santa answers questions only if he is not sleeping, of course!
- When Santa is wakened up by a reindeer, this one must be the last reindeer who just came back. Only the last reindeer can wake up Santa.
- Santa always takes care reindeers first.
- While Santa is attaching sleigh and delivering toys, all reindeers must be there.

Write a C++ program using **ThreadMentor** to simulate these activities. Note that you can only use mutex locks and semaphores. **Your program will not be counted as a correct one if any other synchronization primitives are used.**

Input and Output

The input of your program consists of the following:

- The number of elves ***e***, the number of reindeers ***r***, and the number of toy deliveries ***t*** should be taken from the command line arguments as follows:

  ```
  ./prog4  e  r  t
  ```

  Thus, `./prog4  7  9  8` means Santa's shop has 7 elves and 9 reindeers. After delivering toys 8 times, Santa retires. If ***e*** is 0, the default value is 7; if ***r*** is 0, the default value is 9; and if ***t*** is 0, the default value is 5. For example, `./prog4  0  6  0` means that Santa's shop has 7 elves and 6 reindeers. After delivering toys 5 times, Santa retires. All command line arguments are non-negative.
- Your program should generate an output similar to the following:

  ```
  Santa thread starts
       .....
       Reindeer 3 starts
       .....
            Elf 6 starts.
       .....
       Reindeer 1 returns
            Elf 2 has a problem
            Elf 5 has a problem
            Elf 1 has a problem
       Reindeer 3 returns
            Elves 1, 2, 5 wake up the Santa
  Santa is helping elves
            Elf 7 has a problem
       Reindeer 2 returns
  Santa answers the question posted by elves 1, 2, 5
            Elves 1, 2, 5 return to work
               ..........
       The last reindeer 5 wakes up Santa
  Santa is preparing sleighs
            Elf 4 has a problem
               ..........
  The team flies off (1)!  // the number indicates how many toy deliveries
            Elf 2 has a problem
               ..........
  After (XX) deliveries, Santa retires and is on vacation!
  ```

  Due to the dynamic behavior of multithreaded programs, you will not be able to generate an output that is exactly identical to the above.
- All messages from the Santa thread start on column 1; all messages generated by reindeer threads start on column 5; and all messages printed by elf threads start on column 10. Without observing this

rule, you may risk lower grade.

- The output below illustrates that elves 2, 5 and 1 have problems and form a group. They wake up Santa. Then, Santa answers their questions and release them. After this, the elves go back to work and the Santa goes back to sleep.

```
                    Elf 2 has a problem
                    Elf 5 has a problem
                    Elf 1 has a problem
                    ..........
                    Elves 1, 2, 5 wake up the Santa
                    ..........
          Santa answers the question posted by elves 1, 2, 5
                    ..........
                    Elves 1, 2, 5 return to work
```

- The following illustrates the interaction between reindeers and the Santa. Reindeers return one by one. The last one, reindeer 5 here, wakes up Santa. Then, Santa prepares the sleigh. After this, the team flies off. The number in parenthesis is the number of toy deliveries.

```
               Reindeer 1 returns
               ..........
               Reindeer 3 returns
               ..........
               Reindeer 2 returns
               ..........
               The last reindeer 5 wakes up Santa
               ..........
          Santa is preparing sleighs
               ..........
          The team flies off (1)!
```

- After a specific number of toy deliveries, Santa prints out a message and stops his toy-making operation.

```
          After (XX) deliveries, Santa retires and is on vacation!
```

Submission Guidelines

General Rules

1. All programs must be written in C++.
2. Use the `submit` command to submit your work. You can submit as many times as you want, but only the last on-time one will be graded.
3. Unix filename is case sensitive, `THREAD.cpp`, `Thread.CPP`, `thread.CPP`, etc are ***not*** the same.
4. We will use the following approach to compile and test your programs:

```
          make                      <-- make your program
          ./prog4 e r t             <-- test your program
```

   This procedure may be repeated a number of times with different input files to see if your program works correctly.
5. Your implementation should fulfill the program specification as stated. Any deviation from the specification will cause you to receive **zero** point.
6. A `README` file is always required.

7. **No late submission will be graded.**
8. **Programs submitted to wrong class and/or wrong section will not be graded.**

Compiling and Running Your Programs

This is about the way of compiling and running your program. If we cannot compile your program due to syntax errors, wrong file names, etc, **we cannot test your program, and, as a result, you receive 0 point**. If your program compiles successfully but fails to run, **we cannot test your program, and, again, you receive 0 point**. **Therefore, before submitting your work, make sure your program can compile and run properly.**

1. **Not-compile programs receive 0 point.** By **not-compile**, I mean any reason that could cause an unsuccessful compilation, including missing files, incorrect filenames, syntax errors in your programs, and so on. Double check your files before you submit, since I will **not** change your program. Note again: Unix filenames are ***case sensitive***.
2. **Compile-but-not-run programs receive 0 point. Compile-but-not-run** usually means you have attempted to solve the problem to some degree but you failed to make it working properly.
3. **A meaningless or vague program receives 0 point even though it compiles successfully.** This usually means your program does not solve the problem but serves as a placeholder or template just making it to compile and run.

Program Style and Documentation

This section is about program style and documentation.

1. For each file, the first piece should be a program header to identify yourself like this:

```
// -----------------------------------------------------------
// NAME : John Smith                          User ID: xxxxxxxx
// DUE DATE : mm/dd/yyyy
// PROGRAM ASSIGNMENT #
// FILE NAME : xxxx.yyyy.zzzz (your unix file name)
// PROGRAM PURPOSE :
//    A couple of lines describing your program briefly
// -----------------------------------------------------------
```

   Here, **User ID** is the one you use to login. It is ***not*** your social security number nor your M number.

   For each function in your program, include a simple description like this:

```
// -----------------------------------------------------------
// FUNCTION  xxyyzz : (function name)
//     the purpose of this function
// PARAMETER USAGE :
//    a list of all parameters and their meaning
// FUNCTION CALLED :
//    a list of functions that are called by this one
// -----------------------------------------------------------
```

2. Your programs must contain enough concise and to-the-point comments. Do not write a novel!
3. Your program should have good indentation.

4. Do not use global variables except for semaphores and other absolutely needed entities such as counters!

Program Specification

Your program must follow exactly the requirements of this programming assignment. Otherwise, you receive 0 point even though your program runs and produces correct output. The following is a list of potential problems.

1. Your program does not use the indicated algorithms/methods to solve this problem.
2. Your program does not follow the structure given in the specification. For example, your program is not divided into functions and files, etc when the specification says you should.
3. Any other significant violation of the given program specification.
4. **Incorrect output format.** This will cost you some points depending on how serious the violations are. The grader will make a decision. Hence, carefully check your program output against the required one.
5. **Your program does not achieve the goal of maximum parallelism.**

Program Correctness

If your program compiles and runs, we will check its correctness. We normally run your program with two sets of input data, one posted on this programming assignment page (the public one) and the other prepared by the grader (the private one). You program must deliver correct results for both data sets. Depending on the seriousness of the problem(s), significant deduction may be applied. For example, if your program delivers all wrong results for the public data set, you receive 0 point for that component.

The `README` File

A file named `README` is required to answer the following questions:

- The logic of your program
- Why does your program work?
- Convince me that your program works properly. More precisely, answer the following questions. **Make sure you will have a convincing argument for each question.** Note that argument such as ***because a semaphore is used to ....., the indicated situation cannot happen***" will not be counted as **convincing**. You should explain why the situation will not happen through the use of a semaphore or semaphores. Providing arguments like that will receive low or very low grade.
    1. **Question:** How do you guarantee that only three elves will ask questions?
    2. **Question:** Show that why no elf will leave before the questions are answered.
    3. **Question:** Show that while three elves are waiting for an answer, no other elves can cut in and ask questions.
    4. **Question:** How do you guarantee that Santa only answers question while he is not sleeping.
    5. **Question:** Show that when Santa is wakened up by a reindeer, this reindeer is the last one coming back from vacation.
    6. **Question:** How do you make sure Santa always handles reindeers first.
    7. **Question:** Show that while Santa is attaching the sleigh and delivering toys, all reindeers are there. They won't sneak out for vacation.
    8. **Question:** Show that while Santa is attaching the sleigh and delivering toys, elves will not ask questions.

You should elaborate your answer and provide details. **When answering the above questions, make sure each answer starts with a new line and have the question number (*e.g.*, Question X:) clearly shown. Separate two answers with a blank line.**

Note that the file name has to be `README` rather than `readme` or `Readme`. Note also that there is ***no*** filename extension, which means filename such as `README.TXT` is ***NOT*** acceptable.

`README` must be a plain text file. We do not accept files produced by any word processor. Moreover, watch for very long lines. More precisely, limit the length of each line to no more than 80 characters with the Return/Enter key for line separation. Missing this file, submitting non-text file, file with long lines, or providing incorrect and/or vague answers will cost you many points. Suggestion: Use a Unix text editor to prepare your `README` rather than a word processor.

Final Notes

1. Your submission should include the following files:
    1. File `thread.h` that contains all class definitions of your threads.
    2. File `thread.cpp` contains all class implementations of your threads.
    3. File `thread-support.h` contains all needed definitions of your support functions. See the next bullet.
    4. File `thread-support.cpp` contains all supporting functions such as `AskQuestion()`, `ReindeerBack()`, `WaitOthers()`, `WaitSleigh()`, `FlyOff()`, `Sleep()` and other functions designed and implemented by you as needed.
    5. File `thread-main.cpp` contains the main program.
    6. File `Makefile` is a makefile that compiles the above three files to an executable file `prog4`. **Since we may use any lab machine to grade your programs, your makefile should make sure all paths are correct. Note also that without following this file structure your program is likely to fall into the compile-but-not-run category, and, as a result, you may get a low grade. Before submission, check if you have the proper file structure and correct makefile.** Note that your `Makefile` **should not** activate the visualization system.
    7. File `README`.

    Note also that without following this file structure your program is likely to fall into the compile-but-not-run category, and, as a result, you may get low grade. Therefore, before submission, check if you have the proper file structure and a correct makefile.
2. **Always start early, because I will not grant any extension if your home machine, network connection, your phone line or the department machines crash in the last minute.**
3. **Since the rules are all clearly stated, no leniency will be given and none of the above conditions is negotiable. So, if you have anything in doubt, please ask for clarification.**
4. **Click here to see how your program will be graded.**