School of Electrical and Computer Engineering

CptS223: Advanced Data Structures C/C++

Fall 2019

Programming Assignment 3 (PA3)

Linked List & Array

Due: Wednesday 10/09/2019 @ 11:59pm

0. Instructions

All source code must be in C++. You are required to use the Unix environment. Your submission should accompany a report that explains your work. Your code will be tested on EECS servers. Thus, you are strongly encouraged to test (or event write from the beginning) your code on EECS servers before submitting them. If you do not have an EECS account, you should get one immediately. The EECS IT team will assist you with setting up your EECS account. Look for IT on this page for contact information of the IT personnel:

https://school.eecs.wsu.edu/people/staff/

If you have not logged into EECS Gitlab, you should do so immediately. Information about how to install Git on your computer and how to upload your package (code + report) to the EECS Git can be found under "**Programming Assignments**" section of the course web page. All submission files should be submitted through Git. Note that submissions through any other means for example by emailing your package to the instructor/TA/TAs will be discarded and will NOT be graded. So please follow the above instructions carefully.

You can find the course web page following this link:

http://epsl.eecs.wsu.edu/teaching/

This assignment is an individual programming assignment. No team work is allowed.

The final material for submission should be entirely written by you. If you decide to consult with others or refer materials online, you MUST give due credits to these sources (people, books, webpages, etc.) by listing them in the report that accompanies your submission. Note that no points will be deducted for referencing these sources. However, your discussion/consultation should be limited to the initial design level (if any). Sharing or even showing your source code/assignment solution verbiage to anyone else in the class, or direct reproduction of source code/verbiage from online resources, will all be considered plagiarism, and will therefore be awarded ZERO points and subject to the WSU Academic Dishonesty policy. (Reproducing from the Weiss textbook is an exception to this rule, and such reproduction is encouraged wherever possible.)

Late submission policy: A late penalty of 10% will be assessed for late submissions within the next 24 hours (i.e., until midnight of the next day after the submission deadline). After this one-time late submission, no submissions will be accepted.

1. Description of the Assignment

The goal of this programming exercise is to identify the performance and implementation tradeoffs between the linked list and array ADTs, by implementing the following game.

The MyJosephus problem is the following game: N people, numbered 0 to N-1, are sitting in a circle. Starting at person 0, a single hot potato is passed around. After M passes, the person holding the hot potato is eliminated, the circle closes ranks, and the game continues with the person who was sitting after the eliminated person picking up the hot potato to begin the next round of passing. The game is played until only one person is left (with the potato, of course), and that last remaining person wins (both the game and the potato!). For example, if M=0 and N=5, players are eliminated in order {0,1,2,3}, and player number 4 (i.e., the 5th player) wins; If M=1 and N=5, the order of elimination is {1,3,0,4} before 2 wins.

An animated example for M=2, N=5 is illustrated in the provided powerpoint slides accompanying this assignment. You are encouraged to go over the slides for better understanding of the game.

Your task is to provide two different implementations for the MyJosephus problem using STLs list and vector, respectively. Your program should build upon the following source code:

- **Person.h** encapsulates each player in the game;
- **ListMyJosephus.h** provides the required class interface for the list implementation;
- **VectorMyJosephus.h** provides the required class interface for the vector implementation;
- **ElapsedTimeExample.cpp** is an example code that shows how to calculate processing time in C++. You are also free to use other timing functions like gettimeofday() (just Google "gettimeofday example").

The above-mentioned files are included in this assignment package.

Using the above source code, implement your own **ListMyJosephus.cpp**, **VectorMyJosephus.cpp**, and **Person.cpp**. Feel free to add more functions as needed for your implementation. The above source code is provided to give you a template to start with - i.e., it is by no means complete (or would even compile). You will have to make whatever changes necessary to complete, compile, and getting it working for the project.

Please note that the Vector and List that you will use should always maintain meaningful data about the game and you should perform appropriate List/Vector operations as needed by the game. For example, you cannot perform lazy deletion in your linked list. That is, you cannot mark an element in the linked list (or vector) as "deleted" (by for example maintaining a flag) without actually deleting that element from the list/vector.

For testing and reporting, implement two separate test programs called **TestListMyJosephus.cpp** and **TestVectorMyJosephus.cpp** (not provided here). These programs should implement the following steps:

1. Instantiate an object of the corresponding *MyJosephus class with a user-supplied parameter values for N and M;
2. Play a full game until a winner is found;
3. After each elimination round, output the list of players still left in the game, starting from the lowest player id.
4. At the end of the game, report the elimination sequence and the winner (in a single line);
5. Report the following timing statistics:
    a. total time for playing the game,
    b. average elimination time which is the average for the time spent between two consecutive eliminations.

While recording both these times, do NOT include time for step 3 (which prints the pending list after each round). You can comment out that part of the code while measuring performance. It's there strictly for your debugging reasons.

6. The timing statistics should be in seconds or milliseconds or microseconds (whichever gives the closest precision to measure the actual time of the event). Now, it may so happen that if a particular timed event's time is less than a second, your timer function will show 0 seconds. Obviously this does not mean 0 seconds. It just means you got to measure the time at a lower/finer resolution and use milliseconds or microseconds. If it turns out that the timed event is smaller than a microsecond, then you can consider that time to mean really "0" seconds - i.e., nothing to add to your timer variable.

2. Report

Report Format:

In a separate written document (in Word or PDF), compile sections A through D as follows:

A: Problem statement. In one or two sentences, summarize the goal of the problem.

B: Algorithm design. Within one page, provide a brief description of the main ideas behind your algorithms for the two implementations. If you provide the information about how you have initialized the circular list or array of players at the start of the game, and describe the method to eliminate the next player at each step of the game using the underlying data structure, that will be sufficient. You can use pseudocodes (or plain English) to express this part in the report (use figures if possible).

C: Experimental setup. In this section, you should provide a description of your experiment setup, which includes but is not limited to

- Machine specification.
- How many times did you repeat each experiment before reporting the final timing statistics?
- O/S and environment used during testing: Windows or Unix? Also mention which compiler environment (e.g., g++, Visual Studio). This information will help the TAs determine where to run your programs during grading.

D: Experimental Results & Discussion. In this section, you should report and compare the performance (running time) of the list and vector implementations based on your observations, and provide justification for your observations.

Results:

For your testing and reporting, conduct the following two sets of experiments separately for each of your two implementations:

- Experiment #1: Keep M fixed at 3, and vary N in powers of two starting from 4,8,16,32,.. up to at least 1,024. Go even higher if possible, as long as the overall time is under a few minutes.
- Experiment #2: Keep N fixed (at say 512 or 1,024) and vary M in powers of two starting from 2,4,8,... up to the largest power of 2 less than N.

In your report for the results, address the following points:

- Make 4 plots for each of your two implementations.
    - Plot I) total running time on y-axis vs. N on x-axis;
    - Plot II) average elimination time between on y-axis vs. N on x-axis;
    - Plot III) total running time on y-axis vs. M on x-axis;
    - Plot IV) average elimination time on y-axis vs. M on x-axis;
- You must use some electronic tool (e.g., excel, matlab, gnuplot) to create the plot. Hand-drawn plots will NOT be accepted. The plots should be to scale and pasted into the report. Do not attach any source code that you may have written to generate the plots. If you used excel, you can attach the Excel spread sheet but the plots separately still need to be pasted into your report.

Discussion:

(a) Provide a discussion of your results, which includes but is not limited to:

- Which of the two implementations (list vs. vector) performs the best and under what conditions? Does it depend on the input?
- How does the running time dependency on the parameter N compare with the dependency on the parameter M?

(b) Support all your observations made with solid to-the-point justification. For e.g., are the observations consistent with your theoretical expectations or are they in contrary? If so, why? Any theoretical analysis to help understand and explain the observations or discrepancies conceptually will be the best form of justification. This subsection is very important!

3. Grading

Obviously, this assignment is not just about implementing the Josephus game correctly and getting it working, but is also about exploring two design routes, comparing them both analytically and empirically, and reporting the performance results with justifications. Therefore, the points during grading will be distributed as follows:

Assume that the whole assignment is worth 100 points:

CODING (50 pts):

- (10 pts): Does the code implement the Josephus game correctly?
- (15 pts): Is the code implemented in an efficient way? i.e., are there parts in the code that appear redundant, or implemented in ways that can be easily improved? Does the code conform to good coding practices of Objected Oriented programming?
- (15 pts): Does the code compile and run successfully on a couple of test cases?
- (10 pts): Is the code documented well and generally easy to read (with helpful comments and pointers)?

REPORT (50 pts):

- (15 pts): Is the algorithm designed efficiently? Are appropriate data structures used (where choice is provided)? Does the algorithm exploit the advantages of the different data structures used in an effective manner?
- (5 pts): Experimental setup specified.
- (10 pts): Are results shown as plots in a well annotated manner and do the general trend of results make sense?
- (20 pts): Are the justifications/reasons provided to explain the observations analytically sound? Is there a reasonable attempt to explain anomalies (i.e., results that go against analytical expectations), if any?

Obviously to come up with the above evaluation, the TAs are going to both read and run your code.