High Performance Computing

Homework #2: Part B

Due: Tue February 16 2016 by 11:59 PM (Midnight)

Email-based help Cutoff: 5:00 PM on Mon, Feb 15 2016

Maximum Points For This Part: 20

Submission Instructions

This homework assignment must be turned-in electronically via Canvas. Ensure your C++ source code is named with the convention `MUid_hw2.cpp` (example: `raodm_hw2.cpp`), where *MUid* is your Miami University unique ID. You may suitably organize your C++ source code into as many additional methods as needed. Upload just your source code onto Canvas. **Do not submit archive files such as: zip,7zip,rar,tar,tar.gz,rpm,deb etc.**

Objective

The objective of this homework is to develop a basic C++ program to implement a frequently used gather operation in a memory efficient manner.

Grading Rubric:

This is an advanced course and consequently the expectations in this course are higher. Accordingly, the program submitted for this homework must pass necessary tests in order to qualify for earning a full score.

NOTE: Program that do not compile, have methods longer than 25 lines, skeleton code, or hard coded solutions will be assigned zero score.

Scoring for this assignment will be determined as follows assuming your program compiles (and is not skeleton code):

- Since the objective is to be memory efficient, solutions that use intermediate strings (or buffers) to solve this problem (Using integer or character variables is fine but not arrays, strings, vectors, etc) will be assigned only a partial score:
  - Undergraduates: Max 15 points (if solution is not memory efficient).
  - Graduates: Max of 12 (if solution is not memory efficient)

- **-1 Points**: for each warning generated by `g++` when compiling your C++ program with the `-Wall` (show all warnings) flag.

- **-1 Points**: for each warning generated by the CSE departments' C++ style checker (a slightly relaxed version from Google Inc). On Ben 002 lab machines you can download the style checker from Canvas→Files→Handouts. You can run the C++ style checker as shown below:

```
[homework2]$ chmod +x cpplint.py
[homework2]$ cpplint.py raodm_hw2.cpp
Done processing raodm_hw2.cpp
Total errors found: 0
```

- **NOTE:** Points will be deducted for violating stylistic qualities of the program such as: program follows formatting requirements (spacing, indentation, suitable variable names with appropriate upper/lowercase letters, etc.). The program includes suitable comments at appropriate points in each method to elucidate flow of thought/logic in each method. Program strives to appropriately reuse as much code as possible.

Homework Exercise

This homework exercise expects you to implement the following gather operation in a memory efficient manner — given a string of size n (n > 2) with k ($0 < k < n$) digits (`'0'`…`'9'`), the gather operation should place all digits at a given index i ($0 \le i < n\text{-}k$).

Memory efficiency: In order for the program to be memory efficient, the solution should not use intermediate strings (or buffers) to solve this problem. Using integer or character variables is fine but you cannot use arrays, strings, vectors, etc. This does increase the time complexity (to $O(n^2)$) at the compromise of memory efficiency.

Starter Code:

In order to streamline this homework the following file(s) are supplied for this homework:

i. `hw2.cpp`: This file includes empty method that is to be developed as part of this homework. This file also includes a `main` method to facilitate testing of your code. Ensure the `main` method remains unmodified when you submit your program for grading.

Additional information & Tips

Gather is a common operation that is applicable in many scenarios where selections or information that is spread out in a list is gathered together starting at a given location while maintaining their relative orders. Note that in computer science algorithms that maintain relative order of elements are called "stable" algorithms. In this exercise, you are expected to develop a stable gather operation. For example given the string `str = "12ab3cde4"`, `gather(str, 3)`, modifies `str` by gathering all digits starting at the final index position of 3 resulting in modifying `str` to `"abc1234de"`.

Tips:

1. In order to develop a memory efficient version of the gather operation, approach the problem as involving a series of swap operations (see `std::swap`).
2. Use `std::isdigit` method to detect if a given character is a digit (`'0'...'9'`).
3. Consider splitting the problem into two parts, one part dealing with digits before and second part dealing with digits after the given gather index position.
4. Conceptually, the problem is not complicated. However, getting the bounds and swap operations can be time consuming. Use the debugger in `NetBeans` to trace through your program to help you troubleshoot your logic / implementation.
5. Utilize several methods to develop your solution.

Sample Outputs

Expected outputs from multiple independent runs of the completed program are shown below. User inputs are shown in red for improved readability

```
$ ./raodm_hw2
Enter gather position (-1 to quit) and string:
5 ab1cdefgh
Gathered @ 5: abcde1fgh
Enter gather position (-1 to quit) and string:
6 1abcdefgh
Gathered @ 6: abcdef1gh
Enter gather position (-1 to quit) and string:
8 1abcdefgh
Gathered @ 8: abcdefgh1
Enter gather position (-1 to quit) and string:
8 abcdefgh1
Gathered @ 8: abcdefgh1
Enter gather position (-1 to quit) and string:
0 1abcdefgh
Gathered @ 0: 1abcdefgh
Enter gather position (-1 to quit) and string:
-1
```

This is the base case example. Your program must operate correctly for strings with 1 digit to earn any points for this homework.

```
$ ./raodm_hw2
Enter gather position (-1 to quit) and string:
0 12ab3cdefghij456klm7n8op9q0
Gathered @ 0: 1234567890abcdefghijklmnopq
Enter gather position (-1 to quit) and string:
6 12ab3cdefghij456klm7n8op9q0
Gathered @ 6: abcdef1234567890ghijklmnopq
Enter gather position (-1 to quit) and string:
16 12ab3cdefghij456klm7n8op9q0
Gathered @ 16: abcdefghijklmnop1234567890q
Enter gather position (-1 to quit) and string:
-1
```

Testing for grading:

The test for grading will be accomplished using the supplied input data file and the given reference output file.

```
$ ./raodm_hw2 < hw2_test_input.txt > my_output.txt
$ diff my_output.txt hw2_ref_output.txt
$
```

If your solution is operating as expected, the above `diff` command will not generate any output (indicating your outputs are identical to the expected output).

Style checking:

Ensure you program meets the stipulated CSE style guidelines using the cpplint.py style checker. You can download the C++ style checker program (same one used during laboratory exercises) from Canvas→`Files`→`Handouts` folder.

Turn-in:

Submit just the C++ source file that meets the requirements of this part of the homework to Canvas. No credit will be given for submitting code that does not compile or just skeleton code. Verify that your program meets all the requirements as stated in the grading rubric. Ensure your C++ source files are named with the stipulated naming convention. Upload all the necessary C++ source files to onto Canvas. Do not submit zip/7zip/tar/gzip files. Upload each source file independently.