CPSC457 Winter 2021 - Assignment 1

Due date is posted on D2L.
Individual assignment. Group work is NOT allowed.
Weight: 15% of the final grade.

Warning: This is a very simple assignment. Please do not assume that subsequent assignments will be this simple.

Motivation for this assignment:

- Well written Python code can run faster than badly written C/C++ code.
- Common reason for bad performance is using unnecessary system calls.
- In this assignment you improve the performance of a badly written C++ program.

Start by cloning the GIT repository:

```
$ git clone https://gitlab.com/cpsc457/public/a1
$ cd a1
```

The repository contains:

`palindrome.py`	Python 3 program that reads in text from standard input and reports the longest palindrome to standard output.
`slow-pali.cpp`	Not a very good C++ version of palindrome.py. Feel free to re-use any part of this code in your solution.
`Makefile`	Makes compilation a bit easier.
`t1.txt … t5.txt`	Some test files.
`dup.py`	Python3 script that can generate big data (see appendix).

For this assignment we will use the following definitions:

Standard input	Please read http://www.linfo.org/standard_input.html .
White space	Whatever `isspace()` reports as white-space. Read the man page for `isspace()` for more information.
Word	Non-zero-length sequence of characters delimited by white space.
Palindrome	Any word that remains the same after reversing it and ignoring the case. Examples of palindromes: "`Did`", "`01-!-10`", "`x`"
Longest palindrome	If there are multiple palindromes, the longest one is reported. If multiple palindromes have the same maximum length, report the first one.

Q1 - Written question (4 marks)

For this question you will compare the performance of the python program (`palindrome.py`) to the C++ program (`slow-pali.cpp`) by using the `time` Unix utility. To time how long it takes to execute the python program on different test files, for example `t4.txt` and `t5.txt`, execute the following commands:

```
$ time python3 palindrome.py < t4.txt
Longest palindrome: redder
real    0m0.300s
user    0m0.296s
sys     0m0.003s
$ time python3 palindrome.py < t5.txt
Longest palindrome: DetartrateD
real    0m0.023s
user    0m0.016s
sys     0m0.007s
```

Before you can time the C++ program, you need to compile it first. Here is how to compile the C++ code by using the included `Makefile`:

```
$ make
g++ -O2 -Wall slow-pali.cpp -o slow-pali
```

If you wish, you can also compile it by hand:

```
$ g++ -O2 -Wall slow-pali.cpp -o slow-pali
```

Now you can time the resulting executable `slow-pali`:

```
$ time ./slow-pali < t4.txt
Longest palindrome: redder
real    0m2.929s
user    0m1.477s
sys     0m1.450s
$ time ./slow-pali < t5.txt
Longest palindrome: DetartrateD
real    0m0.003s
user    0m0.000s
sys     0m0.003s
```

Answer the following questions:

a) Time the python code and C++ code on `t4.txt` and `t3.txt` using the `time` utility. Copy/paste the outputs from the terminal output to your report.
b) How much time did the C++ and python programs spend in kernel vs user mode?
c) Why is the python program faster on some inputs when compared to C++ code? Why is the python program slower on other inputs?

Q2 - Programming question (15 marks)

It is embarrassing for a C++ run slower than Python. Your job is to improve the code in `slow-pali.cpp` by writing a new implementation called `fast-pali.cpp`. Your new implementation should be faster than `slow-pali.cpp` and at least as fast as `palindrome.py` for all possible inputs!

Requirements

- Your program must read input from standard input.

- You are only allowed to use the `read()` system call wrapper. You cannot use any other APIs, such as `mmap()`, `fopen()`, `fread()`, `fgetc()`, or C++'s streams. Hint: all you need to do is to adjust the code to use `read()` with a buffer size of 1MiB.
- Do not store the entire input in memory. You need to write your code so that it can handle any input size, even if it is bigger than the available memory.
- Your program must run on `linuxlab.cpsc.ucalgary.ca`. Use SSH to test your code before you submit it!

Valid input

Your program must be able to handle any text input of up to 2GiB in size. You may assume that no word will be longer than 1024 bytes.

Some test files are available to you, but it is expected that you create your own test files to help you validate your solutions. Your TAs will grade your code on inputs that are not published to you. Only valid inputs will be used to grade your code.

Marking

Your code needs to be both correct, and efficient. Programs that output wrong results, or run very slowly, will receive 0 marks. On 2Gib input your program should finish under 30s on linuxlab machines:

```
$ ./dup.py 2000000000 < t4.txt | time ./fast-pali
Longest palindrome: redder
28.79user 0.25system 0:29.52elapsed 98%CPU (0avgtext+0avgdata 4076maxresident)k
0inputs+0outputs (0major+382minor)pagefaults 0swaps
```

Here is another test you can run:

```
$ seq 101010101 | time -p ./ fast-pali
Longest palindrome: 100000001
real 10.18
user 9.87
sys 0.09
```

The D2L page for this assignment may contain additional hints, and also common questions and answers.

Q3 - Written question (4 marks)

Measure the performance of your `fast-pali.cpp` from Q2 and compare it to the timings you obtained for `slow-pali.cpp` and `palindrome.py` in Q1. Answer the following:

a) Is your `fast-pali.cpp` faster than `slow-pali.cpp`? Why do you think that is?
b) Is your program faster or slower than `palindrome.py` and why?

Justify your answers by using the `strace` utility with the `"-c"` command line option. Include the output of `strace` in your report.

Submission

Submit two files for this assignment to D2L:

1. Answers to the written questions Q1 and Q3 combined into a single file called `report.[pdf|docx|txt]`. Do not use any other file formats.
2. Your solution to Q2 in a file called `fast-pali.cpp`.

Submit these files as two separate files, i.e. **do not** submit an archive, such as ZIP or TAR. If you submit an archive, you will receive a penalty.

General information about all assignments:

1. All assignments are due on the date listed on D2L. Late submissions will not be marked.
2. Extensions may be granted only by the course instructor.
3. After you submit your work to D2L, verify your submission by re-downloading it.
4. You can submit many times before the due date. D2L will simply overwrite previous submissions with newer ones. It is better to submit incomplete work for a chance of getting partial marks, than not to submit anything. Please bear in mind that you cannot re-submit a single file if you have already submitted other files. Your new submission would delete the previous files you submitted. So please keep a copy of all files you intend to submit and resubmit all of them every time.
5. Assignments will be marked by your TAs. If you have questions about assignment marking, contact your TA first. If you still have questions after you have talked to your TA, then you can contact your instructor.
6. All programs you submit must run on `linuxlab.cpsc.ucalgary.ca`. If your TA is unable to run your code on the Linux machines, you will receive 0 marks for the relevant question.
7. Unless specified otherwise, you must submit code that can finish on any valid input under 10s on linuxlab.cpsc.ucalgary.ca, when compiled with `-O2` optimization. Any code that runs longer than this may receive a deduction, and code that runs for too long (about 30s) will receive 0 marks.
8. **Assignments must reflect individual work**. For further information on plagiarism, cheating and other academic misconduct, check the information at this link: http://www.ucalgary.ca/pubs/calendar/current/k-5.html.
9. Here are some examples of what you are not allowed to do for individual assignments: you are not allowed to copy code or written answers (in part, or in whole) from anyone else; you are not allowed to collaborate with anyone; you are not allowed to share your solutions with anyone else; you are not allowed to sell or purchase a solution. This list is not exclusive.
10. We will use automated similarity detection software to check for plagiarism. Your submission will be compared to other students (current and previous), as well as to any known online sources. Any cases of detected plagiarism or any other academic misconduct will be investigated and reported.

Appendix – dup2.py utility (aka. testing your code on big files)

Many of you probably do not have enough quota to store 2Gib test files in your accounts. I created a python scripts called `dup.py` to make it possible to test your program on large inputs, without having to store big files.

`dup2.py` is a simple python program that accepts a single command line argument "N", which indicates the number of bytes that the script will generate on standard output. `dup.py` reads in data from stdin, byte by byte, and outputs the data to stdout. It always outputs N bytes. If the data on stdin is bigger than N bytes, only first N bytes are copied. If the data on stdin is shorter than N, the script will repeat the input data, until N bytes are generated. Example:

```
$ echo "hello." | ./dup.py 10
hello.
hel
```

Here is an example of how to feed 2GB of data to your program, generated by repeating t3.txt:

```
$ ./dup.py 2000000000 < t3.txt | ./fast-pali
```

Here is how you can time your code on the same data:

```
$ ./dup.py 2000000000 < t3.txt | time ./fast-pali
```

Here is how to run strace in combination with dup.py:

```
$ ./dup.py 2000000000 < t3.txt | strace -c ./fast-pali
```