PA2 – Hot / Cold data structures

Student Information

Integrity Policy: All university integrity and class syllabus policies have been followed. I have neither given, nor received, nor have I tolerated others' use of unauthorized aid.

I understand and followed these policies: (•) Yes ( ) No

Name: Brian Bauman

Date: 04/20/2019

Submission Details

Final ***Changelist*** number: 351677

Verified build: (•) Yes ( ) No

Number Tests Passed: 6

Required Configurations: DEBUG & RELEASE

Discussion (What did you learn):

Never really knew how consequential the actual layout of a struct/class was in regards to memory alignment and processing speed. Wild stuff. Or how much physical proximity of the data mattered. Do higher-level languages optimize this for you?? Seems so simple that they must, to some extent, right?

Verify Builds

- Follow the Piazza procedure on submission
  - Verify your submission compiles and works at the changelist number.
- Verify that only MINIMUM files are submitted
  - No – Generated files
    - *.pdb, *.suo, *.sdf, *.user, *.obj, *.exe, *.log, *.pdb, *.db, *.user
    - Anything that is generated by the compiler should not be included
  - No – Generated directories
    - /Debug, /Release, /Log, /ipch, /.vs
- Typical files project files that are required
  - *.sln, *.cpp, *.h
  - *.vcxproj, *.vcxproj.filters, CleanMe.bat

Standard Rules

Submit multiple times to Perforce

- Submit your work as you go to perforce several times (at least 5)
  - As soon as you get something working, submit to perforce
  - Have reasonable check-in comments
    - Points will be deducted if minimum is not reached

Write all programs in cross-platform C++

- Optimize for execution speed and robustness
- Working code doesn't mean full credit

Submission Report

- Fill out the submission Report
  - No report, no grade

Code and project needs to compile and run

- Make sure that your program compiles and runs
  - Warning level ALL …
  - NO Warnings or ERRORS
    - Your code should be squeaky clean.
  - Code needs to work "as-is".
    - No modifications to files or deleting files necessary to compile or run.
  - All your code must compile from perforce with no modifications.
    - Otherwise it's a 0, no exceptions

Project needs to run to completion

- If it crashes for any reason…
  - It will not be graded and you get a 0

No Containers

- NO STL allowed {Vector, Lists, Sets, etc...}
  - No automatic containers or arrays
  - You need to do this the old fashion way - ***YOU EARNED IT***

Leave Project Settings

- Do NOT change the project or warning level
  - Any changing of level or suppression of warnings is an integrity issue

Simple C++

- No modern C++
  - No Lambdas, Autos, templates, etc…
  - No Boost
- NO Streams
  - Used fopen, fread, fwrite…
- No code in MACROS
  - Code needs to be in cpp files to see and debug it easy
- ***Exception:***
  - implicit problem needs templates

Leaking Memory

- If the program leaks memory
  - There is a deduction of 20% of grade
- If a class creates an object using new/malloc
  - It is responsible for its deletion
- Any ***MEMORY*** dynamically allocated that isn't freed up is ***LEAKING***
  - Leaking is ***HORRIBLE***, so you lose points

No Debug code or files disabled

- Make sure the program is returned to the original state
  - If you added debug code, please return to original state
- If you disabled file, you need to re-enable the files
  - All files must be active to get credit.
  - Better to lose points for unit tests than to disable and lose all points

No Adding files to this project

- This project will work "as-is" do not add files…
- Grading system will overwrite project settings and will ignore any student's added files and will returned program to the original state

UnitTestConfiguration file (if provided) needs to be set by user

- Grading will be on the UnitTestConfiguration settings
  - Please explicitly set which tests you want graded… no regrading if set incorrectly

Due Dates

- See Piazza for due date and time
- Submit program perforce in your student directory assignment supplied.
- Fill out your this ***Submission Report*** and commit to perforce
  - ***ONLY*** use Adobe Reader to fill out form, all others will be rejected.
  - Fill out the form and discussion for full credit.

Goals

- Learn
  - Data cache / Alignment
  - Hot / Cold data structures
- Understand firsthand how alignment and data cache affects performance

Assignments

1. ***Alignment*** *Identifying data layout and alignment for supplied data structures(C++ classes)*
   - Rework the data structures in ***ReworkData.h***
     - Rearrange the data layout to make the size smaller
     - Explicitly name any padding in the structure
       - char pad0; // for example
   - Create a Print function to show alignment
     - Update the ***Align::PrintMe(…)*** function to print the alignment
     - Use Trace::out(…) to display the data layout and padding

```
data A:
  0x00: 00 00 00 00
  0x04: 00 aa aa aa
-----------------------
size : 8  padding : 3
```

     - Needs to visually show the padding and alignment
       - Total number of bytes
       - Number of padding bytes
     - Mimic the KeenanSampleOutput_Debug.txt
   - No Templates or Boost allowed
     - Use only simple C++ (classes and methods)
     - You cannot leak memory

2. ***HotCold*** *(Rework the supplied linked list data structure to a hot / cold data structure)*
   - Refactoring any necessary conversion/find functions to the linked list
   - Converting the existing data structure data to this new format
   - Feel free to add helper methods
   - No Templates or Boost allowed
     - Use only simple C++ (classes and methods)
     - You cannot leak memory
   - Verify that new data format is the equivalent to the original data structure
   - Profile the before and after performance numbers of the linked list for the given input.

Coding:

- Write all programs in cross-platform C++.
  - Optimize for execution speed and robustness.
- Refactor programming files in your student directorye
  - Student directory
    - /PA2 /…
  - Make sure that program can be compiled and run through the checked in solution
- More details for HotCold problem
  - You need to implement 3 functions:
    - ***Bloated::FindKey()*** - Find a data node in the Bloated data structure
      - You need to search through the nodes using Linked List protocols (next/prev)
      - Failure to use pointers and Next when searching – 0 credit
    - ***HotCold::FindKey()*** - Find a data node in the NEW Hot/Cold data structure
      - You need to search through the nodes using Linked List protocols (next/prev)
      - Failure to use pointers and Next when searching – 0 credit
    - ***HotCold(Bloated *p)*** - Convert from bloated to Hot/Cold data structures
      - Keep your Real-Time conversion from Bloated to HotCold fast (timing is part of the grade)

Results:

- I included my timings in KeenanSampleOutput_Debug.txt and KeenanSampleOutput_Release.txt
  - You can see my timings, for reference.
  - Your timing will vary depending on your machine, but the ratios should indicate how much you improved the performance.
- Interesting results:
  - Hot/Cold (convert): 74.09 ms
- My original timing to find a specific data structure in the code is:
  - Bloated (find): 9.68 ms
  - Hot/Cold (find): 0.52 ms
  - Ratio: 18.36 times faster!!!
    - Cache does yield performance improvements.
  - You might think this is not much,
    - Most games run at 30Hz, so you have 33.33ms to do your whole game per tick.
    - If you are at 60 Hz, you have 16.66ms.
    - Reducing timing from 8 ms to 0.5 ms is quite significant.

Validation

Simple checklist to make sure that everything is submitted correctly

- Is the project compiling and running without any errors or warnings?
- Does the project run ***ALL*** the unit tests execute without crashing?
- Is the submission report filled in and submitted to perforce?
- Follow the verification process for perforce
  - Is all the code there and compiles “as-is”?
  - No extra files
- Is the project leaking memory?

Hints

Most assignments will have hints in a section like this.

- Do many little check-ins
  - Iteration is easy and it helps.
  - Perforce is good at it.
- Look at the lecture notes!
  - A lot of good ideas in there.
  - The code in the examples work.
- For the Alignment
  - The hardest problem might be the printing
  - Do printing in function - Walk and print each byte, (byte by byte)