Basics 3 – Pointers

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: kaijun he

Date: 01/27/2019

Submission Details

Final ***Changelist*** number: 331532

Verified build: (•) Yes ( ) No

Number Tests Passed: 5

Required Configurations: DEBUG only

Discussion (What did you learn):

from this basic assignment figure out how to cast pointer and passing address to get the value stored in that address, have a deep understanding of relationship of the value and the address where value stored.

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

- C++ pointers
    - Saving the world - one dereference at a time.
    - Increasing C++ knowledge and understanding

Assignments

- General:
    - Add code to the body of the functions:
        - Students_PointerWalk()
        - Students_Casting()
    - Run the Unit Tests to verify progress / success
        - 5/5 is the best for this program

- Students_PointerWalk()
    - Code up the pointer test from class (See Below)
        - Literally, type in the exam!
            - Copy the C++ code of the pointer test into the file
            - No hard code answers… just the test
        - Then step through the code
            - Look at the debug windows
            - Look at the memory window
        - Verify with break points and memory windows
            - This is for your benefit.
                - Please do so...
    - This is for you to SEE the code and understand the pointer accessing

- Students_Casting()
  - Understand the three structures, Cat, Bird, and Dog.
  - Understand how they are added arranged inside the ***petStore*** structure.
    - Pay particular attention to the padding and alignment
  - Code the questions 1-19
    - Restrict your answers to the rules/guidelines presented in code
  - Absolutely no harding coding of values
    - Yes - you can see the tests and answers…
      - The goal is to access the answers with C++ code statements.
  - You should be able to answer those questions by paper first
    - Then verify with the code.
    - Make sure you understand these questions / relationships.

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.

- This is pretty easy Basic assignment
  - It is mainly here to help you single step through your code and understand pointers layouts and access commands.
  - The casting section, allows you to access parts of a complicated structure with casting.
    - Note the data is the same, but the way you access changes.
- I expect this assignment to be completed quickly for most of the students
  - Please make sure you fully understand this code without a debugger.
  - Many little lessons here for those who put in the effort.
- Something similar in the exam – Enjoy

Pointer Test / Keenan

Assume that we are working on a LITTLE endian processor
unsigned char data[];
Memory Dump (values in Hex)

```
data =     0x0000: 0xEB, 0xCD, 0x22, 0x4F,
           0x0004: 0x73, 0xB5, 0xF3, 0x35,
           0x0008: 0x23, 0x24, 0x01, 0xFE,
           0x000C: 0xCD, 0xE3, 0x44, 0x85,
           0x0010: 0x66, 0x43, 0x75, 0x33,
           0x0014: 0x39, 0x5C, 0x22, 0x11,
           0x0018: 0x56, 0xA8, 0xAA, 0x13,
           0x001C: 0x64, 0x82, 0x68, 0x26,

unsigned char  *p;  // char are 8-bits wide
unsigned int   *r;  // ints are 32-bits wide
unsigned short *s;  // shorts are 16-bits wide
```

```
p = &data[0];                          Expected output
printf("%x\n",  *(p+3)  );     1)_____________________________
printf("%x\n",  *(p+5)  );     2)_____________________________
p = p + 12;
printf("%x\n",   *(p)  );      3)_____________________________
printf("%x\n",   p[2]  );      4)_____________________________
printf("%x\n",   *p++  );      5)_____________________________
p += 6;
printf("%x\n",   *--p  );      6)_____________________________
printf("%x\n",   p[5]  );      7)_____________________________
p = p + 2;
printf("%x\n",   *p++  );      8)_____________________________
printf("%x\n",  *(p+3) );      9)_____________________________
p = 5 + p;
printf("%x\n",  *(p++) );     10)_____________________________
printf("%x\n",  *(--p) );     11)_____________________________
```

```
data =     0x0000: 0xEB, 0xCD, 0x22, 0x4F,
           0x0004: 0x73, 0xB5, 0xF3, 0x35,
           0x0008: 0x23, 0x24, 0x01, 0xFE,
           0x000C: 0xCD, 0xE3, 0x44, 0x85,
           0x0010: 0x66, 0x43, 0x75, 0x33,
           0x0014: 0x39, 0x5C, 0x22, 0x11,
           0x0018: 0x56, 0xA8, 0xAA, 0x13,
           0x001C: 0x64, 0x82, 0x68, 0x26,

r = (unsigned int *)&data[0]
printf("%x\n",   *(r)  );        12)_____
printf("%x\n",  *(r+5) );        13)_____
r++;
printf("%x\n",   *r++  );        14)_____
r = r + 2;
printf("%x\n",   r[2]  );        15)_____
r = r + 1;
printf("%x\n",   r[0]  );        16)_____

s = (unsigned short *) r;
printf("%x\n",   s[-2] );        17)_____
s = s - 3;
printf("%x\n",   s[2]  );        18)_____
s += 5;
printf("%x\n",  *(s+3) );        19)_____
printf("%x\n",   *(s)  );        20)_____
p = (unsigned char *) s;
printf("%x\n",  *(p+3) );        21)_____
p += 5;
printf("%x\n",  p[-9] );         22)_____
--p;
printf("%x\n",   p[0] );         23)_____
```