403 lines (346 loc) · 16.4 KB

```
/*******************************************************************************

Simple examples showing (possibly) interesting questions in code as a skill
 development and debugging tool (IDE) familiarity exercise.

Very Short Version:
 Search for #TODO marks with questions and answer them. (Create answer doc.) :)

Short Version:
 The #TODO marks are questions - answer them in order in a lab notes document. As
 you go save your changes and commit to your repo as evidence of your work.

Longer Version (recommended steps):
    1. Put this file in your repo folders and commit with a message like "No changes
       yet" or similar.
    2. Open this file in your IDE that *must* support debugging. (VS, CLion etc)
    3. Build and run this file. Make sure this works before going on.
    4. Create a lab-notes document for your answers. (Paste the questions provided
       if you want, or discover them as you go.) Save and commit.
    4. Work your way through each of the file #TODO Q. marks in order.
       - Work one section at a time. There are if (false) { ... } blocks to change to
         if (true) get the code to run.
       - Suggest committing your lab notes to your repo as you do each section (minimum).
         This will make a good history of evidence for you!
       - Suggest turning each finished section back to "false" (as it will probably
         speed up compiling time, depending on your setup.)
       - This code file may not change much as you go (simple false/true changes) so you
         may not need to commit it much (depending on what extra changes you decide to make).

There are quite a few questions (~33) but many are fairly simple. There is a simple
 text file with the questions listed if that helps you with your lab notes.

As a top level view, here are the important sections (topics) in main()
```

```
34          1. Warm up. Create a particle, set values, show to screen
35          2. Get a particle with the values we pass to the function
36          3. Set values in a particle that we already have
37          4. Use a pointer to an existing particle
38          5. Array of structs
39          6. Struct pointer with new and delete for memory
40          7. Array of pointers to structs
41
42      Written by Clinton Woodward <cwoodward@swin.edu.au>
43       for COS30031 Games Programming
44
45      This file is for your personal study use only and must not be shared or made
46       publicly available.
47
48      Updates
49       2020-07-05: Cleanup, new questions and comment help.
50
51      ***************************************************************************/
52      #include <iostream>
53
54      using namespace std;
55
56      // Q.1 What is the difference between a struct and a class?
57      struct Particle
58      {
59          int age;
60          int x;
61          int y;
62      };
63
64      // Q.2 What are function declarations?
65      // Tip: Define, but explain *why* or *if* they are needed.
66      //void showParticle(Particle);
67
68      // Q.3 Why are variable names not needed here?
69      // Extra: Could you add variable names? Would that be good?
70      Particle getParticleWith(int age, int x, int y);
71
72      //void setParticleWith(Particle, int, int, int);
73
74      //void showParticleArray(Particle *, int);
75
76      // Q.4 Does your IDE know if this method is used?
77      // If yes - how does it indicate this? (Colour? Tip? Other?)
78      void showParticleArray_2(Particle arr[], int size);
79
80      // Function details - each matches the function declarations at the top
81      void showParticle(Particle p)
82      {
83          cout << "Particle: ";
```

```
84          cout << "(age=" << p.age << "), ";
85          cout << "(x,y)=(" << p.x << "," << p.y << ")" << endl;
86      }
87
88      Particle getParticleWith(int age, int x, int y)
89      {
90          Particle result;
91          result.age = age;
92          result.x = x;
93          result.y = y;
94          return result;
95      }
96
97      void setParticleWith(Particle &p, int age, int x, int y)
98      {
99          p.age = age;
100         p.x = x;
101         p.y = y;
102     }
103
104     void showParticleArray(Particle * p_array, int size)
105     {
106         // We can't ~actually~ pass an array, so ...
107         // we pass a pointer to the first element of the array!s
108         // ... and the length. Which might be wrong.
109         cout << "showParticleArray call ..." << endl;
110         for (int i = 0; i < size; i++) {
111             cout << " - pos=" << i << " ";
112             showParticle(p_array[i]);
113         }
114     }
115
116     /*void showParticleArray_2(Particle arr[], int size)
117     {
118         // Q.23 What is the difference between this function signature and
119         //  and the function signature for showParticleArray?
120
121         cout << "showParticleArray_2 call ..." << endl;
122
123         // Q.24 Uncomment the following. It gives different values to those we saw before
124         // So it won't work as a way to determine array size - but why?
125
126         if (true) {
127             cout << "Array as arr[] ..." << endl;
128             cout << " - sizeof entire array? " << sizeof(arr) << endl;
129             cout << " - sizeof array element? " << sizeof(arr[1]) << endl;
130             cout << " - array size n is: " << (sizeof(arr) / sizeof(arr[0])) << endl;
131         }
132
133         // NOTE: The above might get warnings (good!). Not all compilers/IDEs though
```

```
133         // NOTE: The above might get warnings (good!). Not all compilers/IDEs though.
134         // Extra: Make a note about what is giving you warnings if you know.
135
136         // This is the same behaviour as original function
137         for (int i = 0; i < size; i++) {
138             cout << " - pos=" << i << " ";
139             showParticle(arr[i]);
140         }
141
142         //return to main for Q.25 ...
143     }*/
144
145     // Main loop. Stuff happens here ...
146     int main()
147     {
148         // 1. Warm up. Create a particle, set values, show to screen
149         if (false) {
150             cout << " << Section 1 >>" << endl;
151
152             Particle a {.age = 0};
153             // Q.5 un-initialised values ... what this show and why?
154             cout << "Q.5: a with partially initialized values ? ... ";
155             showParticle(a);
156
157             // Q.6 Did this work as expected?
158             a.age = 0;
159             a.x = 10;
160             a.y = 20;
161             cout << "Q.6: a with assigned values 0,10,20 ? ... ";
162             showParticle(a);
163
164             // Q.7 Initialisation list - do you know what are they?
165             // Quicker then setting each part of the particle as above!
166             // Do you know about them? If not, find out and make extra notes in your report.
167             // Yes this is a simple question! :)
168             // Your IDE might help suggest what the values are
169             Particle b {0,0,0};
170             cout << "Q.7: b with initialised values 0,0,0 ? ... ";
171             showParticle(b);
172         }
173
174         // 2. Get a particle with the values we pass to the function
175         //    (When you are up to this section, change false to true. Keeps things compact)
176         if (false) {
177             cout << " << Section 2 >>" << endl;
178             Particle p1 = getParticleWith(1,1,3);
179             cout << "Q.8: p1 with 1,1,3 ? ... ";
180             showParticle(p1); // Q.8 Should show age=1, x=1, y=2. Does it?
181
182             p1 = getParticleWith(-1,2,3);
```

```
183             cout << "Q.9: p1 with -1,2,3 ? ... ";
184             showParticle(p1); // Q.9 Something odd here. What and why?
185             // hint: debug, inspect and look at data type details ...
186         }
187 
188         // 3. Set values in a particle that we already have
189         if (false) {
190             cout << " << Section 3 >>" << endl;
191             // This compiles/runs, but ...
192             Particle p1 = {1,1,1};
193             setParticleWith(p1, 5,6,7);
194             cout << "Q.10: b with 5,6,7 ? ... ";
195             showParticle(p1); //Q.10 showParticle(p1) doesn't show 5,6,7 ... Why?
196             // hint: step-into functions with debugger and inspect values (and addresses)...
197         }
198 
199         // 4. Use a pointer to an existing particle
200         if (false) {
201             cout << " << Section 4 >>" << endl;
202             Particle *p1_ptr;
203             // set b to be something sensible
204             Particle p1 = getParticleWith(5,5,5);
205             cout << "p1 with 5,5,5 ? ... ";
206             showParticle(p1);
207             // get address of b, keep it ...
208             p1_ptr = &p1;
209             cout << "Address of p1:" << &p1 << endl;
210             cout << "Value of p1_ptr:" << p1_ptr << endl;
211 
212             // Note that (*p1_ptr).age gets the p1.age value, so ...
213             cout << "Q.11 and Q.12: Test results ..." << endl;
214             if ((*p1_ptr).age == p1.age) cout << " - TRUE!"; else cout << " - False";
215             cout << endl;
216             // Note that (*p1_ptr).age is the same as p1_ptr->age
217             if ((*p1_ptr).age == p1_ptr->age) cout << " - TRUE!"; else cout << " - False!";
218             cout << endl;
219             // Extra: Does C++ have a ternary operator? If so, replace the two if lines above.
220             // Q.11 So what does -> mean (in words)?
221             // Q.12 Do we need to put ( ) around *p1_ptr?
222             // Tip: State what it means, or what it would mean if we didn't write it.
223 
224             // pass the dereferenced pointer as argument
225             cout << "Q.13: p1 via dereferenced pointer ... ";
226             showParticle((*p1_ptr));
227             // Q.13 What is the dereferenced pointer (from the example above)?
228 
229             // update p1, ...
230             p1 = getParticleWith(7,7,7);
231             // Note: p1 is now a new particle struct with new values. So, ...
232             // Q.14 Is p1 stored on the heap or stack?
```

```
233             // Q.15 What is p1_ptr pointing to now? (Has it changed?)
234             // Tip: Use your IDE inspector to check the "address" of p1 and value of p1_ptr
235             cout << "values of new p1 ? ... ";
236             showParticle(p1);
237             cout << "particle values at p1_ptr ?... ";
238             showParticle((*p1_ptr));
239             cout << "address of p1_ptr " << p1_ptr << endl;
240             // Q.16 Is the current value of p1_ptr good or bad? Explain
241
242         }
243         // Q.17 Is p1 still available? Explain.
244
245         // 5. Array of structs
246         if (false) {
247             cout << " << Section 5 >>" << endl;
248             // Q.18 <deleted - ignore> :)
249
250             // NOTE: plain old array - not a fancy std::array
251             // NOTE: zero 0 indexed arrays. (No bounds checking ... probably.)
252             Particle p_array1[3];
253             p_array1[0] = getParticleWith(1,2,3);
254             p_array1[1] = getParticleWith(4,5,6);
255             p_array1[2] = getParticleWith(7,8,9);
256
257             // Q.19 Uncomment the next code line - will it compile?
258             // p_array1[3] = getParticleWith(0,0,0);
259             // - If it compiles, does it run without errors?
260             // Q.20 Does your IDE tell you of any issues? If so, how?
261             // NOTE: Recommend you re-comment the line - it's not needed later
262
263             // show that we can access one element of the array
264             cout << "p_array[1] with 4,5,6 ... ";
265             showParticle(p_array1[1]);
266             // Array of pointers to structs
267             showParticleArray(p_array1, 3);
268             // Q.21 MAGIC NUMBER?! What is it? Is it bad? Explain!
269
270             // Can we work out the length? Yes, but ...
271             cout << "Q.22: Array length?" << endl;
272             cout << " - sizeof entire array? " << sizeof(p_array1) << endl;
273             cout << " - sizeof array element? " << sizeof(p_array1[0]) << endl;
274             cout << " - array size n is: " << (sizeof(p_array1) / sizeof(p_array1[0])) << endl;
275             // Q.22 Explain in your own words how the array size is calculated.
276             // Tip: find out what the sizeof operator is. (It's not a function.)
277             // Q.23-24 Go to the showParticleArray 2 implementation and see there ...
278             cout << "Q.23 and Q.24: showParticleArray_2 differences ..." << endl;
279             showParticleArray_2(p_array1, 3); // alternative signature
280
281             // Tip: An easy array initialisation approach ... (note: it's not a 2-D array!)
```

```
282             cout << "Tip: easy (~nested) initialisation ... " << endl;
283             Particle p_array2[] = {{1,1,1}, {2,2,2}, {3,3,3}};
284             showParticleArray(p_array2, 3); // works fine
285 
286             // Here we are going to read array positions that we haven't set properly.
287             // Q.25 Change the size argument to 10 (or similar). What happens?
288             // Tip: Note the output values shown. Consider if they make sense.
289             // Extra: You might see some values that we set earlier. Does that make sense?
290             cout << "Q.25: Array position overrun ... " << endl;
291             showParticleArray(p_array2, 10); // <-- change size from 3 to 10
292         }
293 
294         // 6. Struct pointer with new and delete for memory
295         if (false) {
296             cout << " << Section 6 >>" << endl;
297             // Some pointer warm-up ideas. What you expect?
298             cout << "Q.26: Warm up concept checks ... " << endl;
299             Particle *p1_ptr; // points to nothing - does it?
300             cout << " - pointer address (does it?): " << hex << p1_ptr << endl;
301             Particle p1 = {9,9,9}; // a real and initialised Particle variable
302             cout << " - pointer address of p1:" << hex << &p1 << endl;
303             p1_ptr = &p1; // copy the point to the same particle
304             cout << " - pointer value of p1_ptr " << hex << p1_ptr << endl;
305             // Q.26 What is "hex" and what does it do? (url in your notes)
306 
307             // Now lets create a Particle that we only access via a pointers
308             cout << "Q.27 and Q.28: Using new and delete ... " << endl;
309             p1_ptr = new Particle();
310             // Q.27 What is new and what did it do?
311             cout << " - pointer address " << hex << p1_ptr << endl;
312             showParticle((*p1_ptr));
313             cout << " - show via de-referenced pointer ... ";
314             showParticle((*p1_ptr));
315             cout << " - set a value via pointer" << endl;
316             p1_ptr->age = 99;
317             showParticle((*p1_ptr));
318             // Clean up!
319             delete p1_ptr;
320             // Q.28 What is delete and what did it do?
321 
322             cout << "Q.29 Can we still show value at pointer address? (It was deleted, so ...) " <<
323             cout << " - pointer address " << hex << p1_ptr << endl;
324             // Q.29 What happens when we try this? Explain.
325             showParticle((*p1_ptr));
326 
327             cout << "Q.30 nullptr vs NULL vs 0 ... for pointers." << endl;
328             // house keeping - if a pointer isn't valid, set it to nullptr/NULL
329             p1_ptr = nullptr; // You might see old/sample code with NULL or == 0
330             cout << " - pointer address " << hex << p1_ptr << endl;
331             // Zero test?
```

```
332             if (p1_ptr == 0) { cout << " - Yes! p1_ptr == 0" << endl; }
333             // Q.30 So, what is the difference between NULL and nullptr and 0?
334
335             //  Q.31 What happens if you try this? (A zero address now, so ...)
336             // NOTE: There is a difference between "run" and "debug" in most IDEs
337             // NOTE: If you do a simple run (not a debug) with the IDE, you should
338             // normally get a "process finished with exit code 0" message at the end.
339             // If the value given is NOT "0", the program stopped with an error code!
340             // Make sure you know if this is the case. Run the program binary directly
341             // from a terminal to confirm if there is an issue.
342             // Debug will tell you *lot* more!
343             if (false) {
344                 cout << "Q31: ";
345                 showParticle((*p1_ptr));
346             }
347         }
348
349
350         // 7. Array of pointers to structs
351         if (true) {
352             cout << " << Section 7 >>" << endl;
353             int n = 5;
354             Particle *ptr_array[n]; // contains pointers to nowhere so far!
355             cout << "Array of pointers - warmup checks:" << endl;
356             cout << "The (direct/root?) ptr_array value " << ptr_array << endl;
357             cout << "Default ptr array values " << endl;
```

COS30031-2023-103071494 / 05 - Lab - Debugging / task05_struct_ptrs_c_arrays.cpp ↑ Top

Code Blame Raw

```
362             // set each pointer to a safe default
363             for (int i = 0; i < n; i++) {
364                 ptr_array[i] = nullptr;
365             }
366             // show the clean pointer values now ...
367             cout << "Clean ptr_array values " << endl;
368             for (int i = 0; i < n; i++) {
369                 cout << " - ptr_array[" << i << "] value " << hex << ptr_array[i] << endl;
370             }
371             // Q.32 Are default pointer values in an array safe? Explain.
372
373             // Reserve memory for each particle and assign address
374             // Note: These are just structs so think memory not constructor in this case.
375             for (int i = 0; i < n; i++) {
376                 ptr_array[i] = new Particle();
377                 ptr_array[i]->age = i; // Note: Set the age so we can tell if it's working :)
378             }
379             // show each particle value
380             cout << "Show each particle pointed to in the pointer array ..." << endl;
```

```
381             for (int i = 0; i < n; i++) {
382                 cout << " - ";
383                 showParticle((*ptr_array[i]));
384                 // Note: we needed (*ptr_array[i]) to turn pointer into Particle parameter
385             }
386             // Q.33 We should always have "delete" to match each "new".
387             //  - What is the problem if we don't delete, and what is the common name for this?
388             // #TODO: Extra: Your IDE may have tools to help you track memory. Does it?
389             // Cleanup! Can you see what happens if you DON'T do this?
390             if (true) {
391                 for (int i = 0; i < n; i++) {
392                     delete ptr_array[i];
393                     ptr_array[i] = nullptr;
394                     // Q.34 Should we set pointers to nullptr? Why?
395                 }
396             }
397 
398             // Note: if we dynamically created the array (with new), we should clean that up too.
399             // #TODO: Q.35 How do you create an array with new and set the size?
400         }
401 
402         return 0;
403     }
```