

# Release Notes – COS30031 Task 5

## Debugging

Author: Luke Valentino

Student ID: 103024456

Unit: COS30031, Games Programming

GitHub Repository: <https://github.com/LukeValentino138/COS30031-2023-103024456>

Date: 25/08/2023

## Summary:

In this lab I have gone through the given cpp file and answered all questions and modified any relevant code.

## Tasks Completed:

- Analysed cpp file.
- Answered Questions.
- Created Lab report.

## Issues:

- None.

## Answers:

Q.1 [line 55] What is the difference between a struct and a class?

A.1 The difference between a struct and a class is that the default accessibility of a struct is public as opposed to a class which is private.

Q.2 [line 63] What are function declarations?

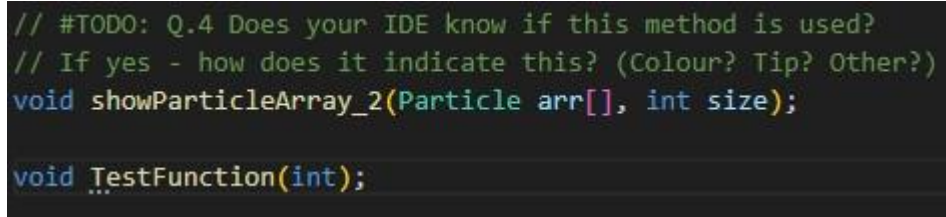
A.2 A function declaration tells the compiler a function's name, return type, and parameters. This allows the functions to be defined in one source file and call it in another.

Q.3 [line 67] Why are variable names not needed here?

A.3 Variables names are not needed here because it is only a function definition. You can add a variable name, but the only benefit is for readability.

Q.4 [line 75] Does your IDE know if this method is used?

A.4 My IDE (Visual Studio Code) does know if the method is used and will underline the function if it is unused. Here is a screenshot to as an example.



```
// #TODO: Q.4 Does your IDE know if this method is used?
// If yes - how does it indicate this? (Colour? Tip? Other?)
void showParticleArray_2(Particle arr[], int size);

void TestFunction(int);
```

Q.5 [line 86] un-initialised values ... what this show and why?

A.5 Visual Studio does underline the variable with the error "Local variable is not initialized". Variables should always be initialized before they are called, otherwise this could lead to unexpected behaviour. When running the program, the x and values are assigned random numbers.

Q.6 [line 95] Did this work as expected?

A.6 Yes it did. The output showed that age = 0, x = 10, and y = 20.

Q.7 [line 97] Initialisation list - do you know what are they?

A.7 In this code, an initialisation list allows for a concise way to set the values Particle b upon initialisation.

Q.8 [line 113] Should show age=1, x=1, y=2. Does it?

A.8 It does.

Q.9 [line 117] Something odd here. What and why?

A.9 The age value was set to -1 in this line "p1 = getParticleWith(-1,2,3);". This creates this abnormality in the output "Q.9: p1 with -1,2,3 ? ... Particle: (age=4294967295), (x,y)=(2,3)". This is because age is an unsigned integer, meaning that it can only be assigned positive numbers.

Q.10 [line 128] showParticle(p1) doesn't show 5,6,7 ... Why?

A.10 The function only modifies its own local copy of p1, to modify the actual p1 you would need to pass Particle by reference.

Q.11 [line 153] So what does -> mean (in words)?

A.11-> means access the member of the object pointed to by.

Q.12 [line 154] Do we need to put ( ) around \*p1\_ptr?

A.12 Parentheses are required when dereferencing a pointer whilst accessing a specific value using ".".

Because the "." operator has a higher precedence than the "8" operator, if we don't use parantheses then the compiler will try to access the age member on p1\_ptr before p1\_ptr is dereferenced. As p1\_ptr itself doesn't have the age member (it just points to something that does), this will not work.

Q.13 [line 160] What is the dereferenced pointer (from the example above)?

A.13 The dereferenced pointer refers to the actual Particle object, in this case p1.

Q.14 [line 165] Is p1 stored on the heap or stack?

A.14 p1 is stored on the stack it was not allocated using "new".

Q.15 [line 166] What is p1\_ptr pointing to now? (Has it changed?)

A.15 It is still pointing to the address of p1. It has not changed.

Q.16 [line 172] Is the current value of p1\_ptr good or bad? Explain

A.16 It seems that the current value of p1\_ptr is good as it is still pointing to p1.

Q.17 [line 175] Is p1 still available? Explain.

A.17 p1 is available if the program is still within that "if (true)" block. As soon as the program exits that block, p1 is no longer available.

Q.18 [line 180] <deleted - ignore> :)

A.18 Ignored

Q.19 [line 189] Uncomment the next code line - will it compile?

A.19 The code is trying to access an out-of-bounds element in array p\_array1. It will compile as C++ does not do bounds checking on native arrays as compile time. The line may or may not throw an error, the behaviour is undefined.

Q.20 [line 192] Does your IDE tell you of any issues? If so, how?

A.20 My IDE does not tell me of any issues. I am using Visual Studio Code.

Q.21 [line 200] MAGIC NUMBER?! What is it? Is it bad? Explain!

A.21 Any number that is seemingly out of place and make the code less readable are typically considered bad code. In this case, the number 3 refers to the size of the array for "showParticleArray" to iterate through.

Q.22 [line 207] Explain in your own words how the array size is calculated.

A.22 Each Particle object consists of three integers (age, x, y). Each integer takes up 4 bytes of space, so each Particle object takes up a total of 12 bytes of space. As the particle array consists of three particle objects, the total size of the array will be 36 bytes.

Q.23 [line 375] What is the difference between this function signature and A.23 Both signatures are functionally the same. When a native array is passed to a function in C++, the function receives a pointer to the first element of the array.

Q.24 [line 380] Uncomment the following. It gives different values to those we saw before

A.24 It will give different values as it is giving the size of the pointer that has been passed to the function and not the size of the array.

Q.25 [line 219] Change the size argument to 10 (or similar). What happens?

A.25 Changing the size argument to 10, the program tries to access elements that are out of bounds. This is array overrun. The random numbers that are outputted are whatever that part of the array just happens to be pointing to. It is not uncommon for this to be previously defined values.

Q.26 [line 237] What is "hex" and what does it do? (url in your notes)

A.26 The term "hex" prints the value in hexadecimal format.

Q.27 [line 242] What is new and what did it do?

A.27 The term "new" is used to allocate memory for an object or variable on the heap.

Q.28 [line 252] What is delete and what did it do?

A.28 The term "delete" is on a pointer to deallocate the memory it points to.

Q.29 [line 256] What happens when we try this? Explain.

A.29 When we try and show the values of a deallocated pointer, it will typically show us garbage values. When a pointer is released, it is best practice to set it to nullptr to avoid use.

Q.30 [line 265] So, what is the difference between NULL and nullptr and 0?

A.30 In C++ the definition of NULL is 0. So functionally, 0 and NULL are the same. They differ from nullptr however, as nullptr is of type "std::nullptr\_t" which can be implicitly converted to any pointer type. nullptr should be used to avoid ambiguity.

Q.31 [line 267] What happens if you try this? (A zero address now, so ...)

A.31 The program will throw an error declaring that p1\_ptr was nullptr.

Q.32 [line 302] Are default pointer values in an array safe? Explain.

A.32 No, default pointer values in an array are not safe. The initial value of the array pointers is indeterminate. Attempting to dereference them without first initialising them can lead to undefined behaviour.

Q.33 [line 317] We should always have "delete" to match each "new".

A.33 If "delete" is not used on each "new", then the program will not deallocate the memory that the "new" occupies. This is memory leak.

Q.34 [line 325] Should we set pointers to nullptr? Why?

A.34 Pointers should always be set to nullptr after using "delete". This ensures that the pointer isn't dangling (containing a reference to a deallocated memory location).

Q.35 [line 330] How do you create an array with new and set the size?

A.35 An example would be:

```
Particle* array_1 = new Particle[10];
```

