COS30031 Ben Holmes 103024841 Lab2 Report: 31/07/2024

## Question 1

If public and private are not specified, struct’s methods and variables are public while classes are private..

## Question 2

A function declaration contains the function's name, return type, and parameters (if any), whereas the definition is the code to be executed

They are needed in order to create functions, the definition can follow immediately after but don’t have to, standard practice is to define the function after the main function has been defined

## Question 3

Variable names aren’t needed here because this is the declaration, where the variables are not directly needed, so only the variable type is required

You can add variable names there and it is a good idea for readability as people looking at your code or you coming back to your code would not know what the variable types for.

## Question 4

No my ide does not know if the method is used, however it can tell me if the method is not defined which is indicated by 3 dots

## Question 5

It outputted a random number that corresponds to the location of the unused memory that was assigned to the values as the variable was uninitialized

## Question 6

Yes it worked as expected

## Question 7

An initializer list is used to directly initialize data members in a class in a constructor and is a quicker method of doing so rather that the classic Datamember = input within the constructor.

No it doesn’t tell me what the values are

## Question 8

Yes it shows the right values

## Question 9

It shows the maximum value an unsigned int can hold. It does this as it is an unsigned int so cannot store the value of -1 or any negative numbers

## Question 10

It updates p which is a copy of p1 not the actual p1 so the values in the main function never change (they have different memory addresses. You would have to return the p value and make p1 = the function or use pointers)

## Question 11

It is an arrow operator which is used to access the members or variables of a structure via a pointer.

## Question 12

The brackets make it so the program does the \*p1\_ptr first, so the pointer is dereferenced before accessing the age variable.

\*p1\_ptr.age is the same as \*(p1\_ptr.age)

## Question 13

A dereferenced pointer is the value that the pointer was pointing at.

## Question 14

P1 is stored on the stack

## Question 15

p1\_ptr is still pointing at p1

## Question 16

The current value of p1\_ptr is good as it is the same as p1 meaning that it is still pointing at p1 which is what it is supposed to do.

## Question 17

It does not compile

## Question 18

It tells me that get particle width is not defined with a red zigzag underline.

## Question 19

A magic number is a raw number in the code (one that is not attached to a variable). It is best not to use magic numbers as it is not clear what they mean and if they need to be changed in multiple places, rather than just changing the value of a global or local variable, you would have to individually change the values in all locations.

## Question 20

Sizeof determines the amount of memory in bytes the variable uses, this is used to determine an arrays length by dividing the overall size of the array by the size of 1 element, which then gives the number of elements.

## Question 21

It passes an array of particles rather than a pointer to a specific particle.

## Question 22

arr is only storing the first particle meaning it has a smaller size than array[0] resulting in the differing values.

## Question 23

A bunch of random values are produced after the original 3 values, some of them are the same as earlier values as the function is just going to the next memory segment and reading the values off that which may include the older values

## Question 24

It points to a particle with random values stored

## Question 25

Hex sets the format flag of the string stream used by cout to hex, meaning it would then output values in hex.

https://cplusplus.com/reference/ios/hex/

## Question 26

It allocates a block of storage of the size of the new object and returns a pointer to that object

## Question 27

It deallocates the block of storage that the pointer is pointing to.

## Question 28

It produces random values once again as there is no longer an object assigned to that block of storage.

## Question 29

NULL is 0 as in an integer with constant value of 0 and a typecast of void\*, nullptr is a prvalue of type nullptr\_t that evaluates to 0

## Question 30

When debugging it throws a read access violation as a result of p1\_ptr is nullptr

## Question 31

No as they are randomly assigned pointers meaning the values could be anything.