Task 05 - Lab Summary Report

Spike: Task_05
Title: Debugging
Author: Thomas Horsley, 103071494

Goals & Deliverables

Aim: To learn to find and fix bugs in code using my IDE's (VSCode) debugging features.

Deliverables:

- Functional, debugged C++ code
- Lab report

Technology, Tools and Resources

Tech and Tools

The project was scripted in Python C++ 17 using the VSCode IDE version 1.76.
UML's and charts are made with *www.Lucidchart.com*

Optionally (though recommended), source control is handled using Git.

VSCode Plugins/Extensions

- C/C++
  Author: Microsoft
  Version: v2023.4.1
- Colorful Comments (I always recommend)
  Author: Parth Rastogi
  Version: 1.0
- Code Runner
  Author: Jun Han
  Version: v0.12.0

Resources

- Echo360 Lectures "*Lecture Title*"
- Title of YouTube video that helped?
  See: https://www.youtube.com/watch?v=axsplPtoQF0&list=LL&index=1&t=409s

Tasks Undertaken

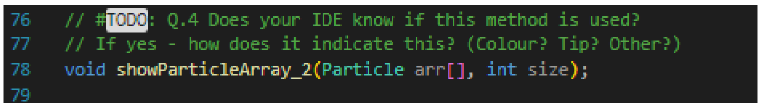
Q & A

1. What is the difference between a struct and a class?
    a. By default, all members of a struct are publicly visible where-as the default visibility of class members is private, other than this a class is nearly identical to a struct. Semantically, the difference between a struct and class lies in their usage in code. Most often a struct will be used to couple similar variables, aiding in data manipulation (think a Vec2 struct as a collection of 2 floats). Additionally, I wouldn't use inheritance on a struct.

2. What are function declarations?
    a. Function declarations are typically housed inside of header (.h) files (along with class variables and Macro definitions) which are referenced by translation units (.cpp files) for their functionality. They allow for the completely independent .cpp files to draw functionality from others. In this case, the use of definitions is unnecessary as all of our functionality lies within the one translation unit.

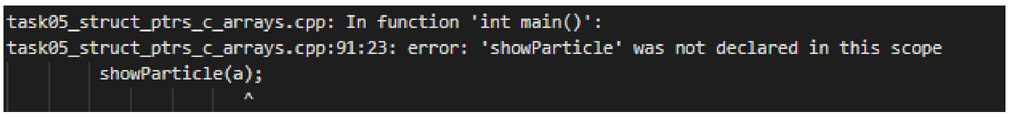
3. Why are variable names not needed here?
    a. Variable names aren't required in definitions as their implementation is ambiguous and variable names can be decided in definition. However, it's good practice to include variable names as easy access descriptors for programmers.

4. Does your IDE know if this method is being used?
    a. Yes, it's known due to the color of the function, unused functions have lower opacity values.



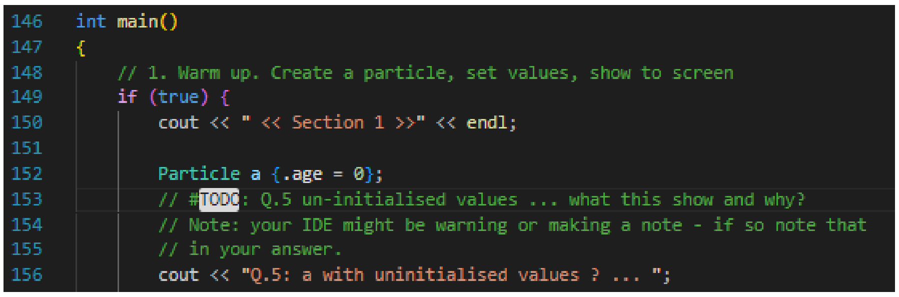
```
// #TODO: Q.4 Does your IDE know if this method is used?
// If yes - how does it indicate this? (Colour? Tip? Other?)
void showParticleArray_2(Particle arr[], int size);
```

5. un-initialized values... what this show and why?
    a. MMMMMMMMM eror.
    Similar to array initializers, the elements of the Particle are unknown if they're uninitialized. They will point to random locations with memory. However, by partially initializing the struct (say age = 0) the rest of the values contained within the struct should default to 0 (shown below).



```
task05_struct_ptrs_c_arrays.cpp: In function 'int main()':
task05_struct_ptrs_c_arrays.cpp:91:23: error: 'showParticle' was not declared in this scope
        showParticle(a);
                      ^
```

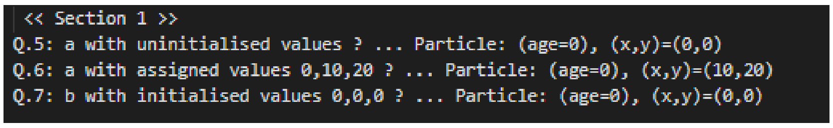
Before partial initialization



```
int main()
{
    // 1. Warm up. Create a particle, set values, show to screen
    if (true) {
        cout << " << Section 1 >>" << endl;

        Particle a {.age = 0};
        // #TODO: Q.5 un-initialised values ... what this show and why?
        // Note: your IDE might be warning or making a note - if so note that
        // in your answer.
        cout << "Q.5: a with uninitialised values ? ... ";
```

Example of partial initialization

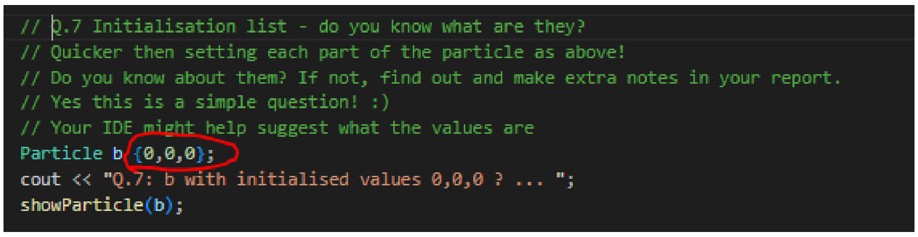


```
 << Section 1 >>
Q.5: a with uninitialised values ? ... Particle: (age=0), (x,y)=(0,0)
Q.6: a with assigned values 0,10,20 ? ... Particle: (age=0), (x,y)=(10,20)
Q.7: b with initialised values 0,0,0 ? ... Particle: (age=0), (x,y)=(0,0)
```

After partial initialization

6. Did this work as expected?
    a. Yes… yes it did.

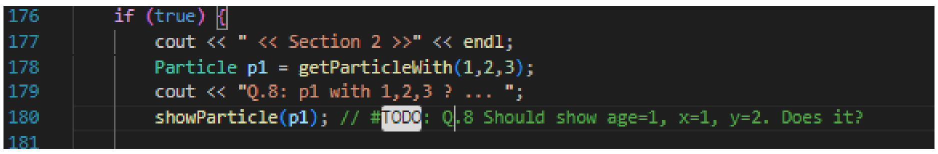
7. Initialization list - do you know what are they?
    a. This initializer list is a concise list containing initial values which is assigned to an object or struct during instantiation.



```
// Q.7 Initialisation list - do you know what are they?
// Quicker then setting each part of the particle as above!
// Do you know about them? If not, find out and make extra notes in your report.
// Yes this is a simple question! :)
// Your IDE might help suggest what the values are
Particle b{0,0,0};
cout << "Q.7: b with initialised values 0,0,0 ? ... ";
showParticle(b);
```

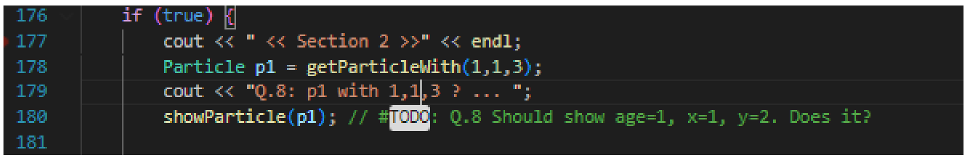
Initialization list circled in red

8. Should show age=1, x=1, y=2. Does it?
    a. It didn't but it does after a parameter change



```
    if (true) {
        cout << " << Section 2 >>" << endl;
        Particle p1 = getParticleWith(1,2,3);
        cout << "Q.8: p1 with 1,2,3 ? ... ";
        showParticle(p1); // #TODO: Q.8 Should show age=1, x=1, y=2. Does it?
```

Original functionality

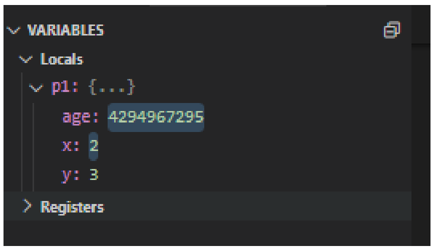


```
    if (true) {
        cout << " << Section 2 >>" << endl;
        Particle p1 = getParticleWith(1,1,3);
        cout << "Q.8: p1 with 1,1,3 ? ... ";
        showParticle(p1); // #TODO: Q.8 Should show age=1, x=1, y=2. Does it?
```

Updated Functionality

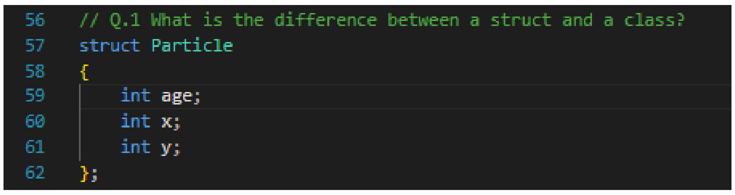
9. Something odd here. What and why?

   a. Given that the value shown in the left figure is $2^{32} - 1$, it's safe to assume this error is caused by the use of an unsigned integer where a signed integer would make sense (though -1 age doesn't make sense but we'll roll with it).



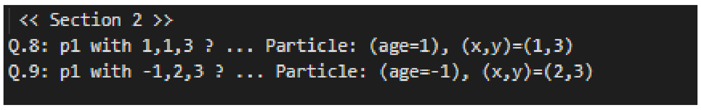
```
p1: {...}
    age: 4294967295
    x: 2
    y: 3
```

Value of *uint* = -1



```
// Q.1 What is the difference between a struct and a class?
struct Particle
{
    int age;
    int x;
    int y;
};
```

Simply change the *unsigned int* parameter to be an *int* and the problem is resolved

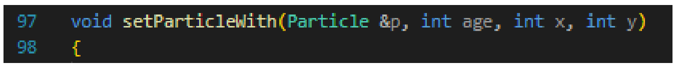


```
 << Section 2 >>
Q.8: p1 with 1,1,3 ? ... Particle: (age=1), (x,y)=(1,3)
Q.9: p1 with -1,2,3 ? ... Particle: (age=-1), (x,y)=(2,3)
```

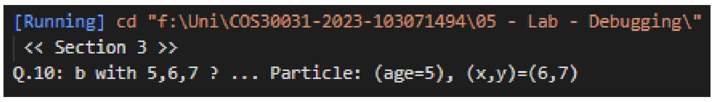
Section 2 final output

10. showParticle(p1) doesn't show 5,6,7... Why?

    a. Initially, the setParticleWith method was taking a Particle struct as a parameter, copying this struct into a new scope and then deleting this new struct once it fell out of scope. What the method was intended to do was take the address of an existing particle on the heap before dereferencing and modifying the particle contained at that address. This issue was fixed by replacing '*Particle p*' with I '*Particle &p*'.



```
void setParticleWith(Particle &p, int age, int x, int y)
{
```



```
[Running] cd "f:\Uni\COS30031-2023-103071494\05 - Lab - Debugging\"
 << Section 3 >>
Q.10: b with 5,6,7 ? ... Particle: (age=5), (x,y)=(6,7)
```

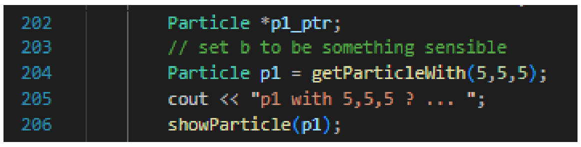
Corrected functionality & output

11. So what does -> mean (in words)?

    a. '- >' is the Member Class Access Operator which allows the programmer to access fields of a class or struct directly and by pointer. Therefore, when working with pointers and modifying data held within a struct or class the '- >' operator must be employed.

12. Do we need to put ( ) around *p1_ptr?
    a. Yes. Without the ( ), the compiler is being asked to used the . (dot) operator to access members of a pointer not a class. The ( ) allow for dereferencing of the pointer.

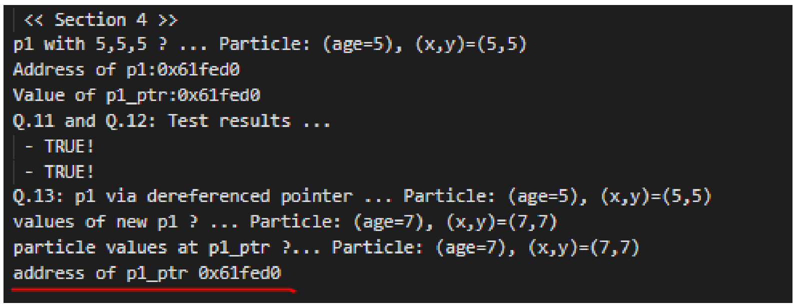
13. What is the dereferenced pointer (from the example above)?
    a. The dereferenced pointer is the initial P1 particle with values 5, 5, 5



```
Particle *p1_ptr;
// set b to be something sensible
Particle p1 = getParticleWith(5,5,5);
cout << "p1 with 5,5,5 ? ... ";
showParticle(p1);
```

14. Is p1 stored on the heap or stack?
    a. p1 is stored on the heap

15. What is p1_ptr pointing to now? (Has it changed?)
    a. p1_ptr is still pointing to the same address it was pointing to at the start of the function

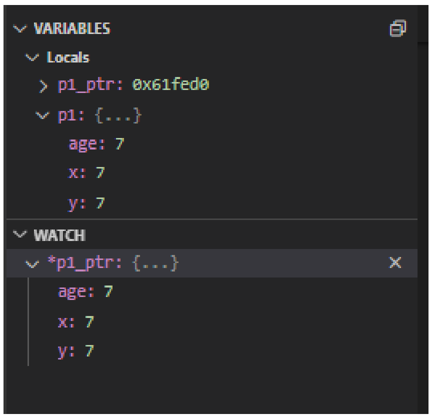


```
 << Section 4 >>
p1 with 5,5,5 ? ... Particle: (age=5), (x,y)=(5,5)
Address of p1:0x61fed0
Value of p1_ptr:0x61fed0
Q.11 and Q.12: Test results ...
 - TRUE!
 - TRUE!
Q.13: p1 via dereferenced pointer ... Particle: (age=5), (x,y)=(5,5)
values of new p1 ? ... Particle: (age=7), (x,y)=(7,7)
particle values at p1_ptr ?... Particle: (age=7), (x,y)=(7,7)
address of p1_ptr 0x61fed0
```

16. Is the current value of p1_ptr good or bad? Explain
    a. p1_ptr is a good pointer as it points to the current p1

17. Is p1 still available? Explain.

    The original p1 no longer exists inside memory as it's address was overridden by the new p1.



```
VARIABLES
  Locals
    p1_ptr: 0x61fed0
    p1: {...}
      age: 7
      x: 7
      y: 7
WATCH
  *p1_ptr: {...}
    age: 7
    x: 7
    y: 7
```

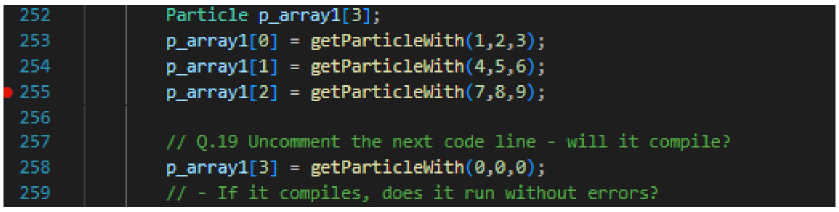
18. <deleted - ignore> :)
    a. Mad

19. Uncomment the next code line - will it compile?

    a. Yes, it compiles and runs

20. Does your IDE tell you of any issues? If so, how?

    a. Yes the program runs without any noticeable issues or crashes, however p_array[3] lies outside of the bounds of our array and can't be accessed. It's probably a security risk in addition to being useless.

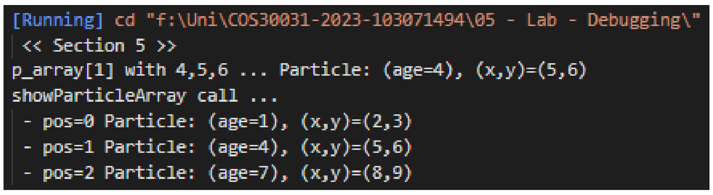


```
    Particle p_array1[3];
    p_array1[0] = getParticleWith(1,2,3);
    p_array1[1] = getParticleWith(4,5,6);
    p_array1[2] = getParticleWith(7,8,9);

    // Q.19 Uncomment the next code line - will it compile?
    p_array1[3] = getParticleWith(0,0,0);
    // - If it compiles, does it run without errors?
```

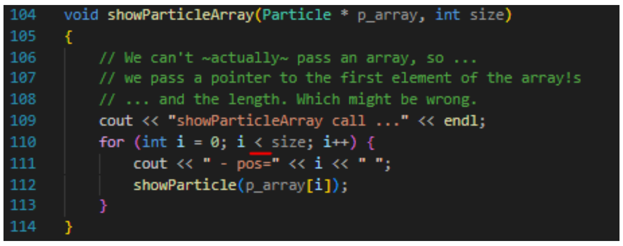
21. MAGIC NUMBER?! What is it? Is it bad? Explain!

    a. It's fine to call as the showParticleArray( ) method takes the size of the array from $0 \text{ to } n-1$.



```
[Running] cd "f:\Uni\COS30031-2023-103071494\05 - Lab - Debugging\"
 << Section 5 >>
p_array[1] with 4,5,6 ... Particle: (age=4), (x,y)=(5,6)
showParticleArray call ...
 - pos=0 Particle: (age=1), (x,y)=(2,3)
 - pos=1 Particle: (age=4), (x,y)=(5,6)
 - pos=2 Particle: (age=7), (x,y)=(8,9)
```

output



```
void showParticleArray(Particle * p_array, int size)
{
    // We can't ~actually~ pass an array, so ...
    // we pass a pointer to the first element of the array!s
    // ... and the length. Which might be wrong.
    cout << "showParticleArray call ..." << endl;
    for (int i = 0; i < size; i++) {
        cout << " - pos=" << i << " ";
        showParticle(p_array[i]);
    }
}
```

showParticleArray functionality

22. Explain in your own words how the array size is calculated.

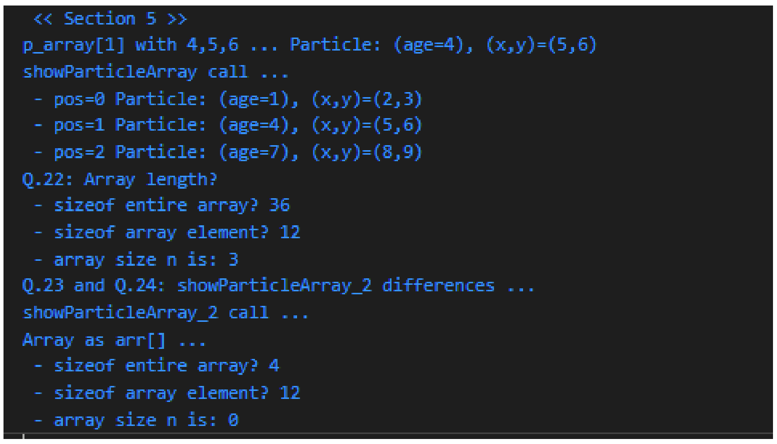
    a. The sizeof( ) operator will return the size of a datatype in bytes. Therefore, to calculate the size of an array (say 32bit / 4byte integers) we take the total size of the array (say 32 bytes) and divide that value by the size of each elements type contained within the array (32/4 = 8 integer long array).

23. What is the difference between this function signature and the function signature for showParticleArray?

    a. showParticleArray took an an array pointer where as showParticleArray_2 takes an array of particles as a parameter

24. Uncomment the following. It gives different values to those we saw before, so it won't work as a way to determine array size - but why?

    a. The output is most likely the result of an integer rounding error as $4/12 < 1$ additionally sizeof( ) is returning the number of elements in the array already so further division is unnecessary.



```
 << Section 5 >>
p_array[1] with 4,5,6 ... Particle: (age=4), (x,y)=(5,6)
showParticleArray call ...
 - pos=0 Particle: (age=1), (x,y)=(2,3)
 - pos=1 Particle: (age=4), (x,y)=(5,6)
 - pos=2 Particle: (age=7), (x,y)=(8,9)
Q.22: Array length?
 - sizeof entire array? 36
 - sizeof array element? 12
 - array size n is: 3
Q.23 and Q.24: showParticleArray_2 differences ...
showParticleArray_2 call ...
Array as arr[] ...
 - sizeof entire array? 4
 - sizeof array element? 12
 - array size n is: 0
```

25. Change the size argument to 10 (or similar). What happens?

    a. The program attempts to read memory which lies outside of the bounds of our array and therefore accesses junk stored in RAM from previous processes.

26. What is "hex" and what does it do?

    a. std::hex allows integer values to be written and read as hexadecimal values.

27. What is new and what did it do?

    a. the *"new"* keyword initializes heap memory for a new variable. In other words, it declares that there will be a variable of type x which requires x space in the heap to exist, then assigns that memory for use.

28. What is delete and what did it do?

    a. Delete does the inverse of new, it will free space in memory and delete variables which we have no use for.

29. What happens when we try this? Explain.

    a. Junk values from memory are outputted as particle ages and coordinates. This occurs as we're trying to read into an uninitialized space in memory.

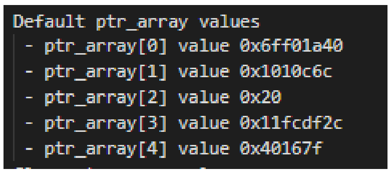
30. So, what is the difference between NULL and nullptr and 0?

    a. nullptr represents a space in memory with address 0, whereas NULL and 0 represent 0 as an integer this can cause ambiguity and is advised against

31. What happens if you try this? (A zero address now, so ...)
    a. This causes a segmentation fault as there's nothing associated with the address.

32. Are default pointer values in an array safe? Explain.
    a. Gonna go with no here chief. Seems exploitable

‘



```
Default ptr_array values
 - ptr_array[0] value 0x6ff01a40
 - ptr_array[1] value 0x1010c6c
 - ptr_array[2] value 0x20
 - ptr_array[3] value 0x11fcdf2c
 - ptr_array[4] value 0x40167f
```

Something just doesn't feel right here

33. What is the problem if we don't delete, and what is the common name for this?
    a. Imagine a situation where I create a new “Table” object, this object will consume x amount of memory when I load it let's say 200MB to be… generous. If an application doesn't clear the memory with this Table object when it isn't being used, it will leak 200MB of memory each time we instantiate a Table eventually resulting in a crash. This is known as a memory leak and why we love garbage collectors.

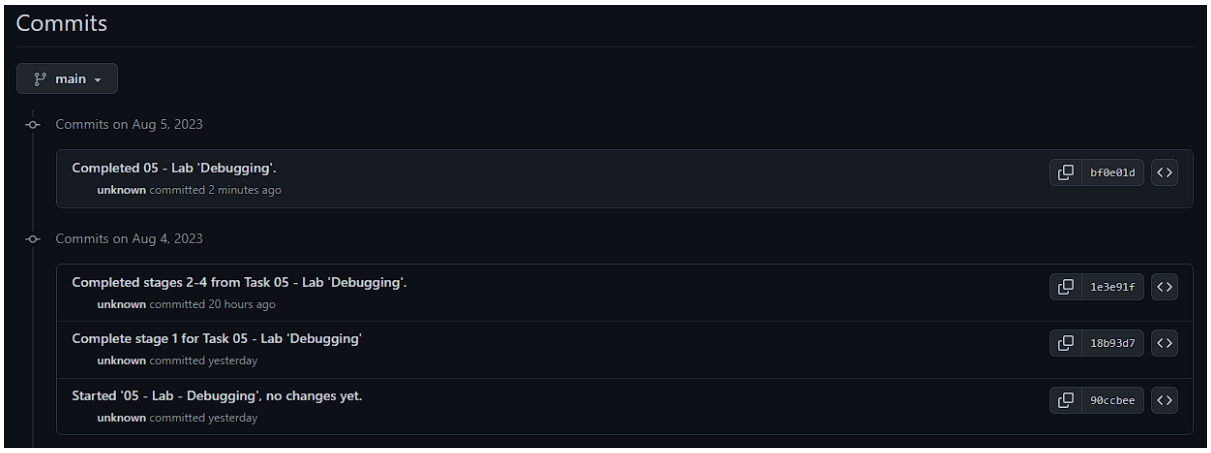
34. Should we set pointers to nullptr? Why?
    a. When deleting pointers, it's considered good practice to set the pointer address to 0. This will avoid double deletes as deleting a pointer pointing to 0 achieves nothing where-as deleting a deleted pointer will cause undefined behavior.

35. How do you create an array with new and set the size?
    a.

```
int main(){
  int size = 5; //the size of a dynamic array can be determined at runtime
  int *dynamicArray = new int[size]; //cool dynamic array

  delete [] dynamicArray;
  return 0;
}
```

Git Commit History



Commits

main

Commits on Aug 5, 2023

Completed 05 - Lab 'Debugging'.
unknown committed 2 minutes ago
bf0e01d

Commits on Aug 4, 2023

Completed stages 2-4 from Task 05 - Lab 'Debugging'.
unknown committed 20 hours ago
1e3e91f

Complete stage 1 for Task 05 - Lab 'Debugging'
unknown committed yesterday
18b93d7

Started '05 - Lab - Debugging', no changes yet.
unknown committed yesterday
90ccbee