

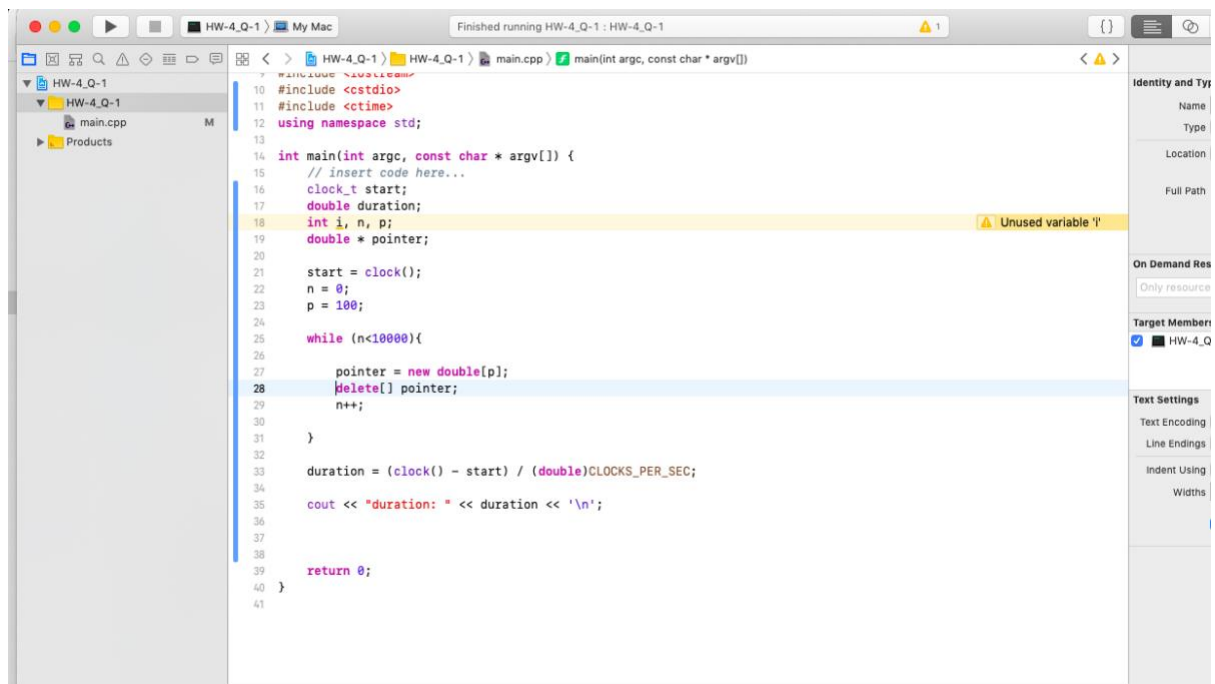
Q1:

Used the Following code for first question , as the array size increases. The code takes more time to execute.

P. ----- Time

100-----duration: 0.001576

1000----duration: 0.001856

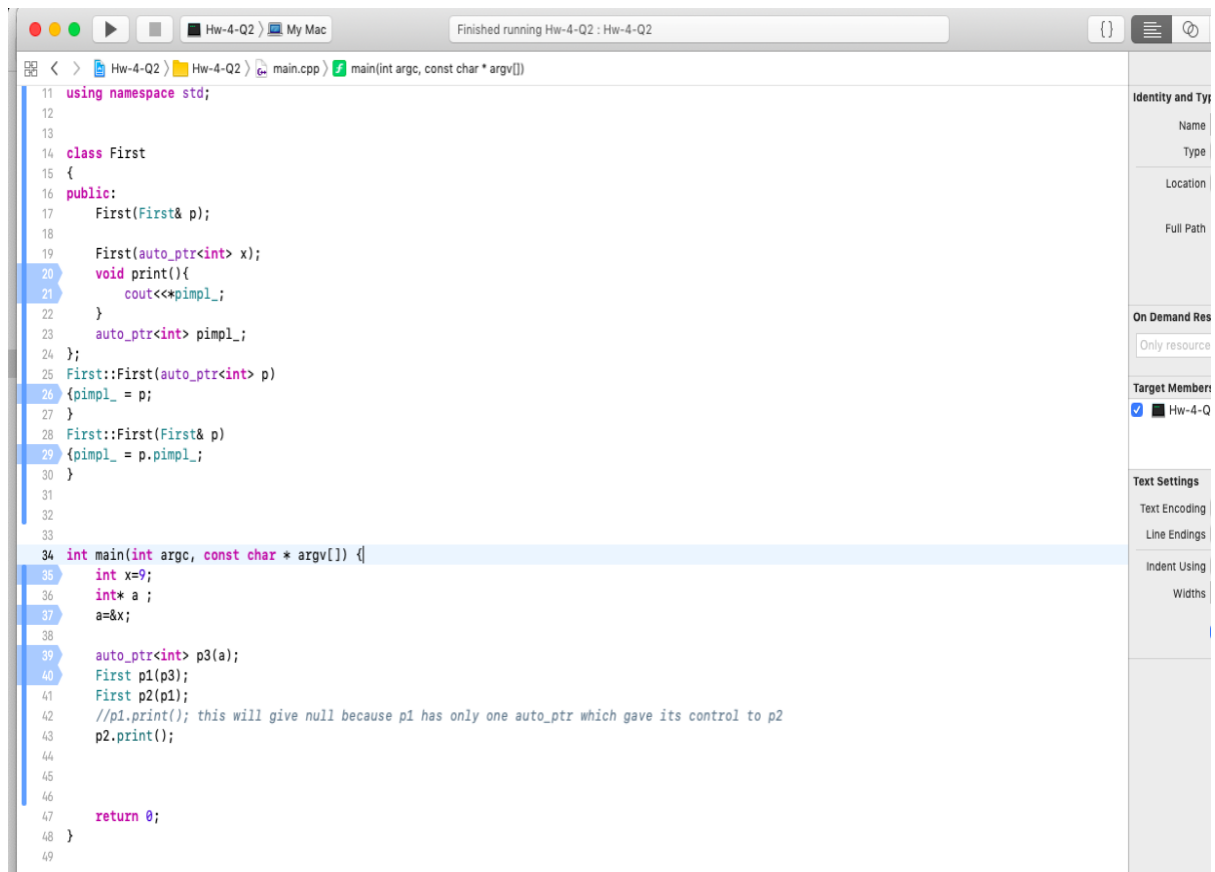


```
10 #include <stdio>
11 #include <time>
12 using namespace std;
13
14 int main(int argc, const char * argv[]) {
15     // insert code here...
16     clock_t start;
17     double duration;
18     int i, n, p;
19     double * pointer;
20
21     start = clock();
22     n = 0;
23     p = 100;
24
25     while (n<1000){
26
27         pointer = new double[p];
28         delete[] pointer;
29         n++;
30     }
31
32     duration = (clock() - start) / (double)CLOCKS_PER_SEC;
33
34     cout << "duration: " << duration << '\n';
35
36
37
38     return 0;
39 }
40
41
```

P.t.o

Q2

- 1.) Its not like normal copy because if you copy auto_ptr p1 to auto_ptr p2 then all the control goes to p2 and p1 points to null. This example gives the idea :



```
11 using namespace std;
12
13
14 class First
15 {
16 public:
17     First(First& p);
18
19     First(auto_ptr<int> x);
20     void print(){
21         cout<<pimpl_;
22     }
23     auto_ptr<int> pimpl_;
24 };
25 First::First(auto_ptr<int> p)
26 {pimpl_ = p;
27 }
28 First::First(First& p)
29 {pimpl_ = p.pimpl_;
30 }
31
32
33
34 int main(int argc, const char * argv[]) {
35     int x=9;
36     int* a ;
37     a=&x;
38
39     auto_ptr<int> p3(a);
40     First p1(p3);
41     First p2(p1);
42     //p1.print(); this will give null because p1 has only one auto_ptr which gave its control to p2
43     p2.print();
44
45
46
47     return 0;
48 }
49
```

The screenshot shows a C++ IDE with a file named `main.cpp`. The code defines a class `First` with two constructors: one taking an `auto_ptr<int>` and another taking a `First` reference. The `print` method outputs the internal `pimpl_` pointer. In the `main` function, an integer `x` is created and its address is stored in `a`. An `auto_ptr<int>` `p3` is created from `a`, and a `First` object `p1` is constructed from `p3`. Then, another `First` object `p2` is constructed from `p1`. A comment indicates that calling `p1.print()` would result in a null pointer because `p1`'s `auto_ptr` control was transferred to `p2` during its construction. Finally, `p2.print()` is called, which successfully prints the value of `x`.

P.t.o

- 2.) You don't need a destructor because auto pointer takes care of freeing that memory area once the encapsulated object goes out of scope & probably not needed anymore because auto pointer takes care of freeing that memory area once the encapsulated object goes out of scope.

Eg:



```
1 //
2 // main1.cpp
3 // Hw-4-Q2
4 //
5 // Created by HIMANSHU KUMAR on 31/10/18.
6 // Copyright © 2018 HIMANSHU KUMAR. All rights reserved.
7 //
8
9 #include <stdio.h>
10 #include <memory>
11 // file c.h
12 //
13 class C
14 {
15 public:
16     C();
17     /*...*/
18 private:
19     class Simple; // forward declaration
20     std::auto_ptr<Simple> pimpl_;
21 };
22
23 // file c.cpp
24 //
25 class C::Simple { /*...*/ };
26
27 C::C() : pimpl_( new Simple ) { }
28
29 //Now the destructor doesn't need to worry about deleting the pimpl_ pointer, because the auto_ptr will handle it automatically.
30
```