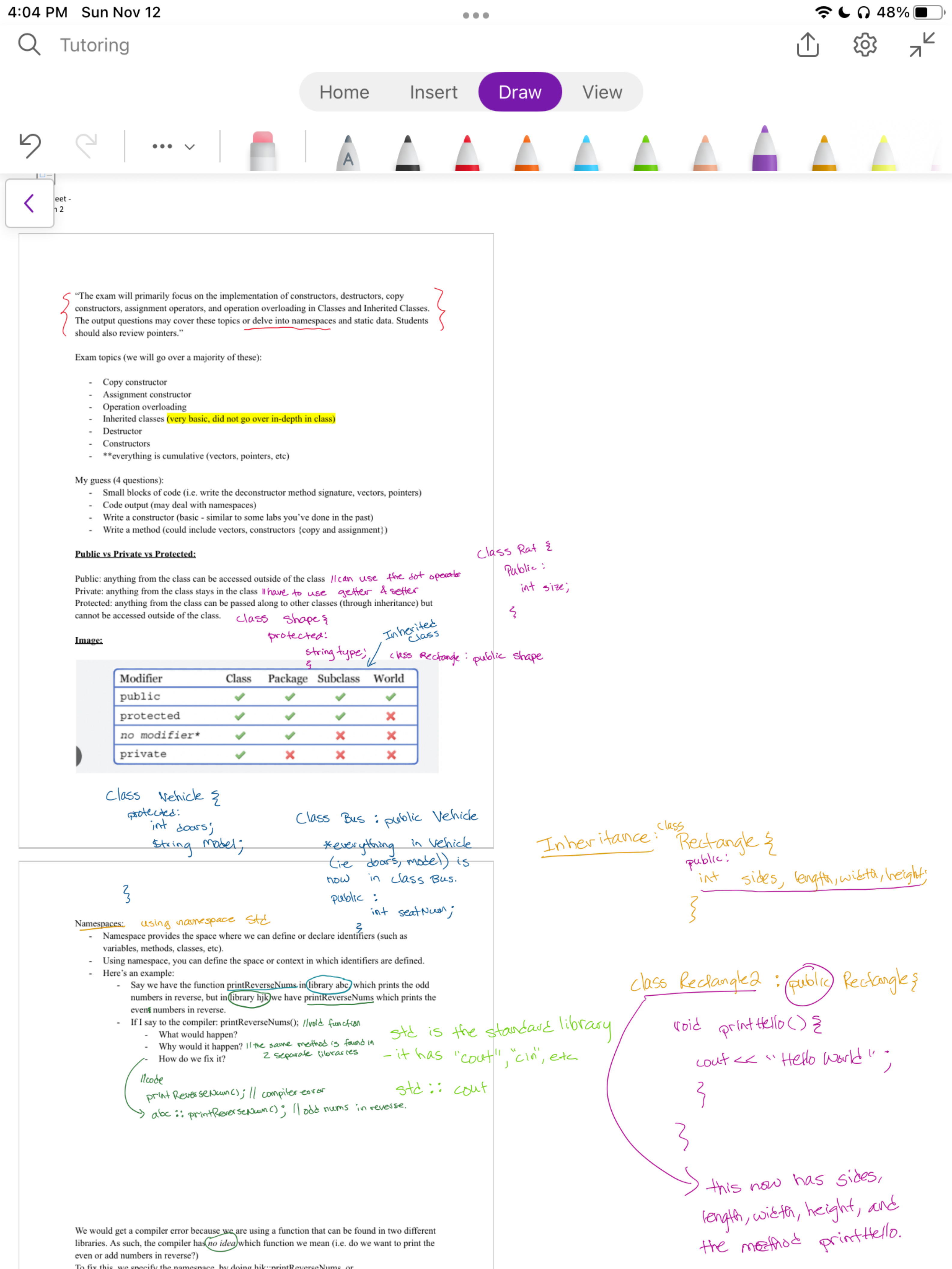


"The exam will primarily focus on the implementation of constructors, destructors, copy constructors, assignment operators, and operation overloading in Classes and Inherited Classes. The output questions may cover these topics or delve into namespaces and static data. Students should also review pointers."

Exam topics (we will go over a majority of these):

- Copy constructor
- Assignment constructor
- Operation overloading
- Inherited classes (very basic, did not go over in-depth in class)
- Destructor
- Constructors
- **everything is cumulative (vectors, pointers, etc)

My guess (4 questions):
- Small blocks of code (i.e. write the deconstructor method signature, vectors, pointers)
- Code output (may deal with namespaces)
- Write a constructor (basic - similar to some labs you've done in the past)
- Write a method (could include vectors, constructors {copy and assignment})

Public vs Private vs Protected:

Public: anything from the class can be accessed outside of the class //can use the dot operator
Private: anything from the class stays in the class //have to use getter & setter
Protected: anything from the class can be passed along to other classes (through inheritance) but cannot be accessed outside of the class.

```
Class Rat {
  Public:
    int size;
}
```

```
class Shape{
  protected:
    string type;
}
```

Inherited class

```
class Rectangle : public shape
```

Image:

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
*no modifier**	✓	✓	✗	✗
private	✓	✗	✗	✗

```
Class Vehicle {
  protected:
    int doors;
    String model;
}
```

```
Class Bus : public Vehicle
  *everything in Vehicle
  (ie doors, model) is
  now in class Bus.
  public :
    int seatNum;
}
```

Inheritance:

```
class Rectangle {
  public:
    int sides, length, width, height;
}
```

```
class Rectangle2 : public Rectangle{
  void printHello() {
    cout << "Hello World";
  }
}
```

this now has sides, length, width, height, and the method printHello.

Namespaces: using namespace std

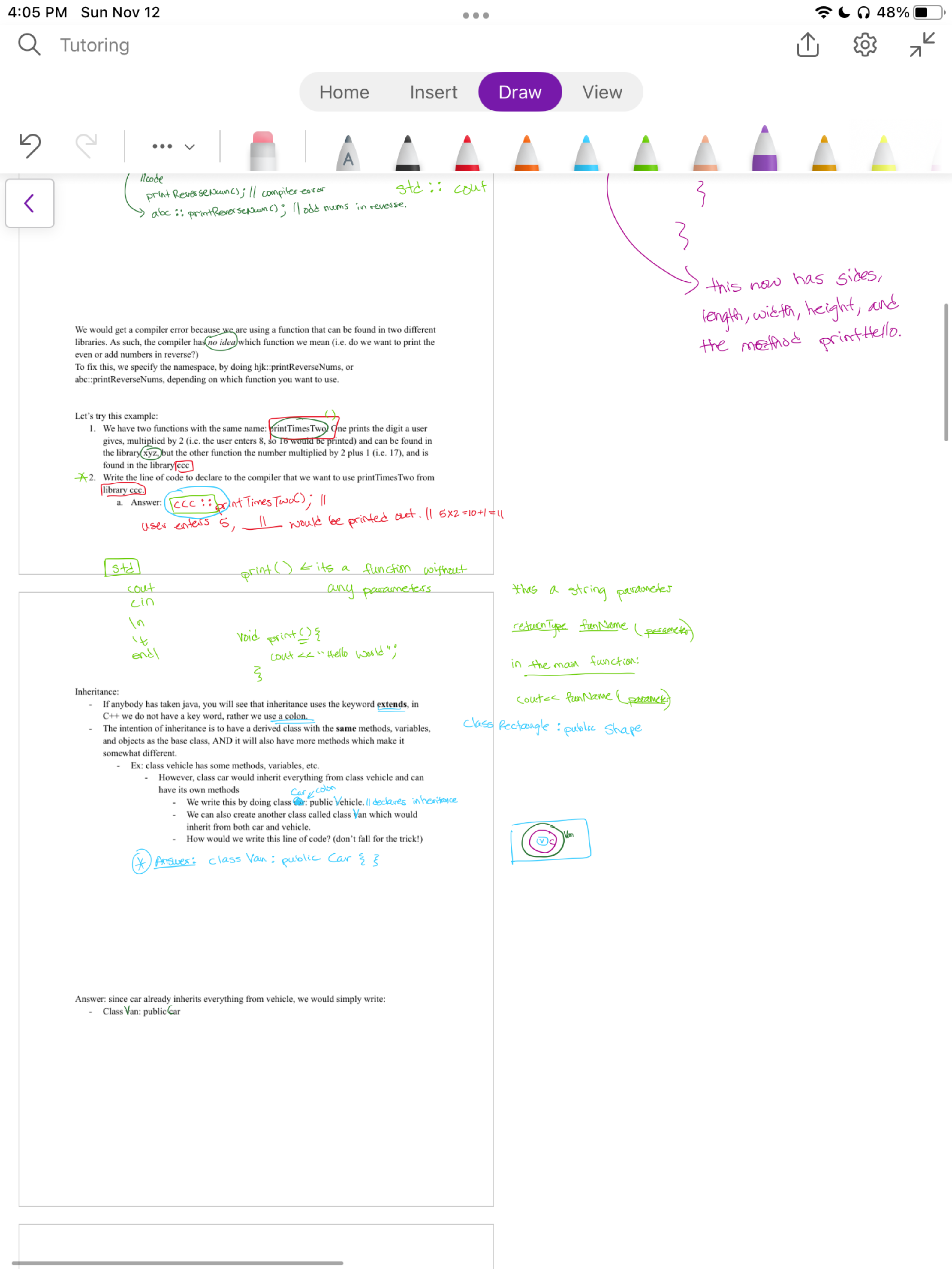
- Namespace provides the space where we can define or declare identifiers (such as variables, methods, classes, etc).
- Using namespace, you can define the space or context in which identifiers are defined.
- Here's an example:
  - Say we have the function printReverseNums in library abc, which prints the odd numbers in reverse, but in library hjk we have printReverseNums which prints the even numbers in reverse.
  - If I say to the compiler: printReverseNums(); //void function
    - What would happen?
    - Why would it happen? //the same method is found in 2 separate libraries
    - How do we fix it?

```
//code
printReverseNum(); // compiler error
abc :: printReverseNum(); // odd nums in reverse.
```

std is the standard library
- it has "cout", "cin", etc

std :: cout

We would get a compiler error because we are using a function that can be found in two different libraries. As such, the compiler has *no idea* which function we mean (i.e. do we want to print the even or add numbers in reverse?)
To fix this, we specify the namespace, by doing hjk::printReverseNums, or



```
//code
printReverseNum(); // compiler error
abc :: printReverseNum(); // add nums in reverse.
```

std :: cout

}

}

this now has sides, length, width, height, and the method printHello.

We would get a compiler error because we are using a function that can be found in two different libraries. As such, the compiler has *no idea* which function we mean (i.e. do we want to print the even or add numbers in reverse?)
To fix this, we specify the namespace, by doing hjk::printReverseNums, or abc::printReverseNums, depending on which function you want to use.

Let's try this example:

1. We have two functions with the same name: printTimesTwo () One prints the digit a user gives, multiplied by 2 (i.e. the user enters 8, so ~~16 would be~~ printed) and can be found in the library xyz, but the other function the number multiplied by 2 plus 1 (i.e. 17), and is found in the library ccc
2. Write the line of code to declare to the compiler that we want to use printTimesTwo from library ccc.
    a. Answer: ccc :: printTimesTwo(); //

user enters 5, 11 would be printed out. // 5x2 =10+1 =11

std

cout
cin
\n
\t
endl

print() ← its a function without any parameters

```
void print(){
    cout << "Hello World";
}
```

*has a string parameter

returnType funName (parameter)

in the main function:

cout<< funName (parameter)

Class Rectangle : public Shape

Inheritance:

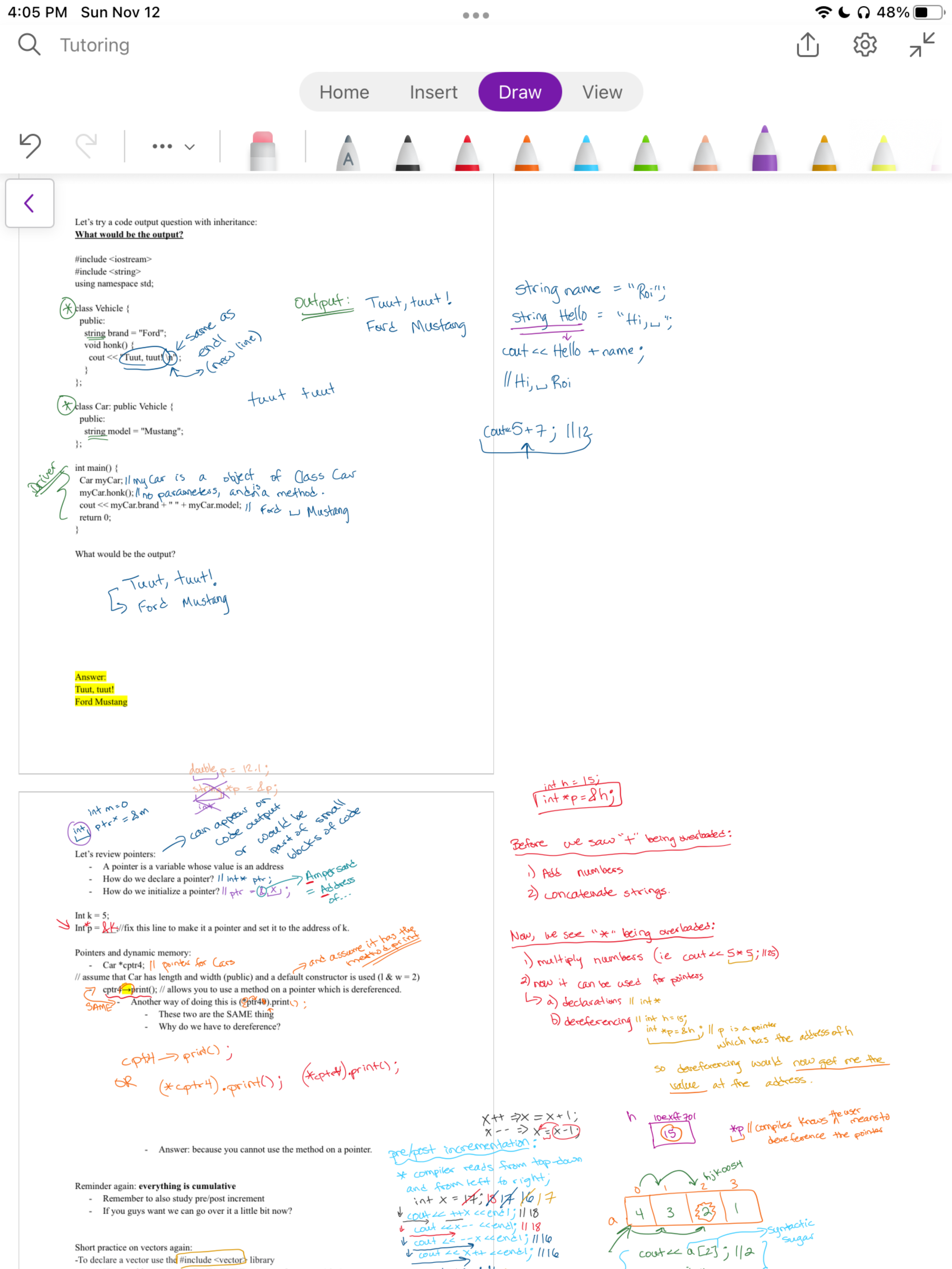
- If anybody has taken java, you will see that inheritance uses the keyword **extends**, in C++ we do not have a key word, rather we use a colon.
- The intention of inheritance is to have a derived class with the **same** methods, variables, and objects as the base class, AND it will also have more methods which make it somewhat different.
    - Ex: class vehicle has some methods, variables, etc.
        - However, class car would inherit everything from class vehicle and can have its own methods
            - We write this by doing class Car: public Vehicle. // declares inheritance
            - We can also create another class called class Van which would inherit from both car and vehicle.
            - How would we write this line of code? (don't fall for the trick!)

Car ↙ colon

(*) Answer: class Van : public Car { }

Answer: since car already inherits everything from vehicle, we would simply write:

- Class Van: public Car



Let's try a code output question with inheritance:

<u>What would be the output?</u>

```
#include <iostream>
#include <string>
using namespace std;

(*) class Vehicle {
  public:
    string brand = "Ford";
    void honk() {
      cout << "Tuut, tuut! \n";     <- same as endl (new line)
    }
};

(*) class Car: public Vehicle {
  public:
    string model = "Mustang";
};

int main() {
  Car myCar; // myCar is a object of Class Car
  myCar.honk(); // no parameters, and is a method.
  cout << myCar.brand + " " + myCar.model; // Ford ␣ Mustang
  return 0;
}
```

Driver

Output: Tuut, tuut!
Ford Mustang

tuut tuut

What would be the output?

Tuut, tuut!
Ford Mustang

Answer:
Tuut, tuut!
Ford Mustang

```
string name = "Roi";
string Hello = "Hi,␣";
cout << Hello + name;
// Hi,␣Roi

Cout<<5+7; //12
```

```
double p = 12.1;
~~string~~ *p = &p;
~~int~~
```

```
int m=0
int ptr* = &m
```

-> can appear on code output or would be part of small blocks of code

Let's review pointers:

- A pointer is a variable whose value is an address
- How do we declare a pointer? // int* ptr;
- How do we initialize a pointer? // ptr = &x; -> Ampersand = Address of...

```
Int k = 5;
Int*p = &k;//fix this line to make it a pointer and set it to the address of k.
```

Pointers and dynamic memory:

- Car *cptr4; // pointer for Cars

// assume that Car has length and width (public) and a default constructor is used (l & w = 2) -> and assume it has the method print

- cptr4→print(); // allows you to use a method on a pointer which is dereferenced.
- SAME Another way of doing this is (*ptr4).print();
    - These two are the SAME thing
    - Why do we have to dereference?

cptr4 -> print();

OR (*cptr4).print(); (*cptr4).print();

- Answer: because you cannot use the method on a pointer.

Reminder again: **everything is cumulative**

- Remember to also study pre/post increment
- If you guys want we can go over it a little bit now?

Short practice on vectors again:

-To declare a vector use the #include <vector> library

```
int h = 15;
int *p = &h;
```

Before we saw "+" being overloaded:

1) Add numbers
2) concatenate strings.

Now, we see "*" being overloaded:

1) multiply numbers (ie cout<< 5*5; //25)
2) now it can be used for pointers
    - a) declarations // int*
    - b) dereferencing // int h = 15; int *p = &h; // p is a pointer which has the address of h

so dereferencing would now get me the value at the address.

```
x++ => x = x+1;
x-- => x = x-1;
```

h 10exff701 [15]

*p // compiler knows the user means to dereference the pointer

pre/post incrementation:

* compiler reads from top-down and from left to right;

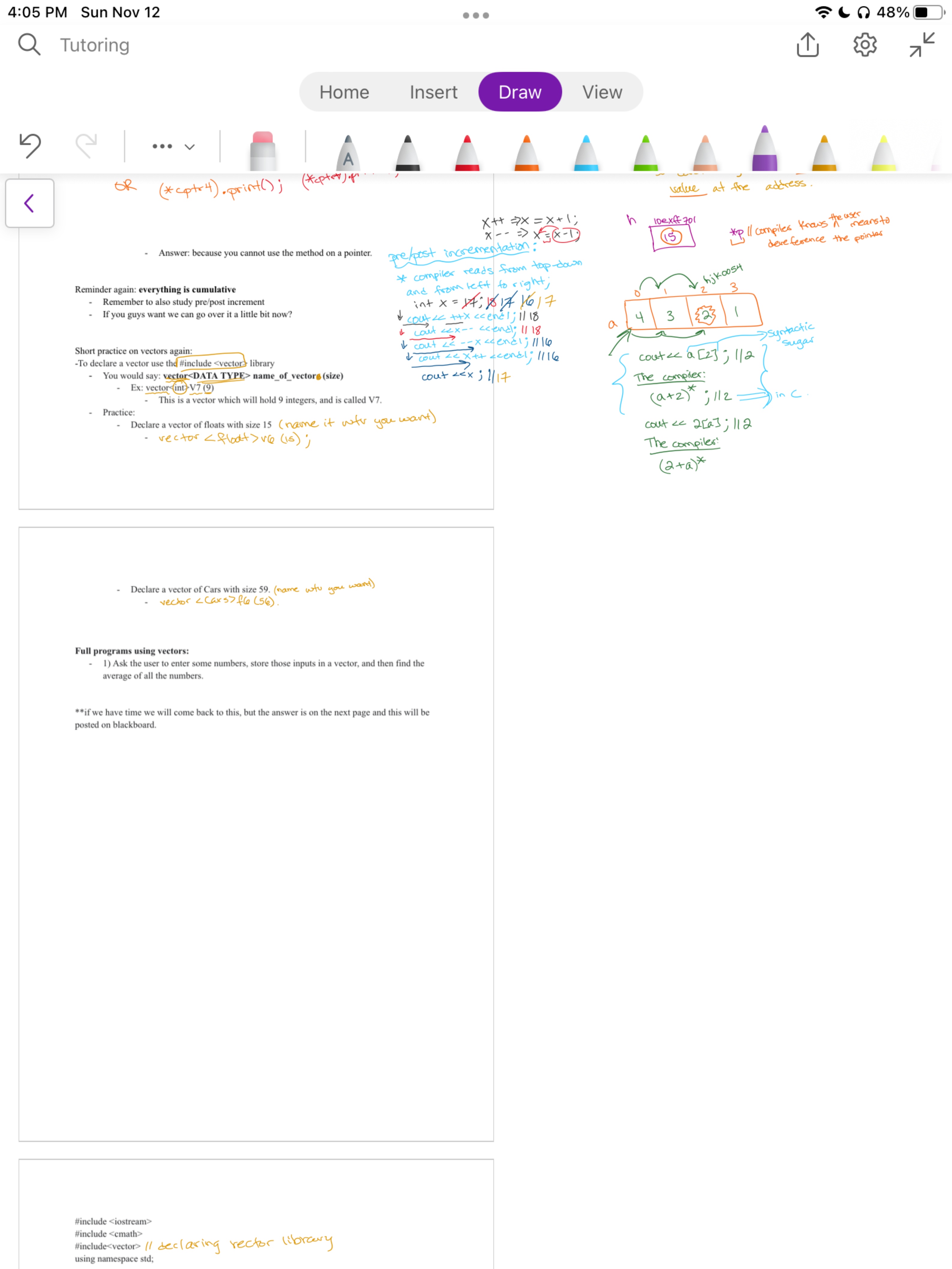
```
int x = 17; 18 17 16 17
cout << ++x << endl; // 18
cout << x-- << endl; // 18
cout << --x << endl; // 16
cout << x++ << endl; // 16
```

hjk0054

0	1	2	3
4	3	2	1

a

cout<< a[2]; //2 -> syntactic sugar



OR (*cptr4).print(); (*cptr4).pr

- Answer: because you cannot use the method on a pointer.

Reminder again: **everything is cumulative**
- Remember to also study pre/post increment
- If you guys want we can go over it a little bit now?

Short practice on vectors again:
-To declare a vector use the #include <vector> library
- You would say: **vector<DATA TYPE> name_of_vectors (size)**
  - Ex: vector<int> V7 (9)
    - This is a vector which will hold 9 integers, and is called V7.
- Practice:
  - Declare a vector of floats with size 15 (name it wtv you want)
    - vector <float> v6 (15);

X++ ⇒ X = X + 1;
X-- ⇒ X = (X - 1);

pre/post incrementation:
* compiler reads from top-down and from left to right;
int x = ~~17~~; ~~18~~ ~~17~~ ~~16~~ 17
cout << ++x << endl; // 18
cout << x-- << endl; // 18
cout << --x << endl; // 16
cout << x++ << endl; // 16
cout << x ; // 17

value at the address.

h
10exff701
15

*p // compiles knows the user means to dereference the pointer

hjk0054
0 1 2 3
a 4 3 2 1

syntactic sugar

cout << a[2]; // 2
The compiler:
(a+2)* ; // 2
in C.

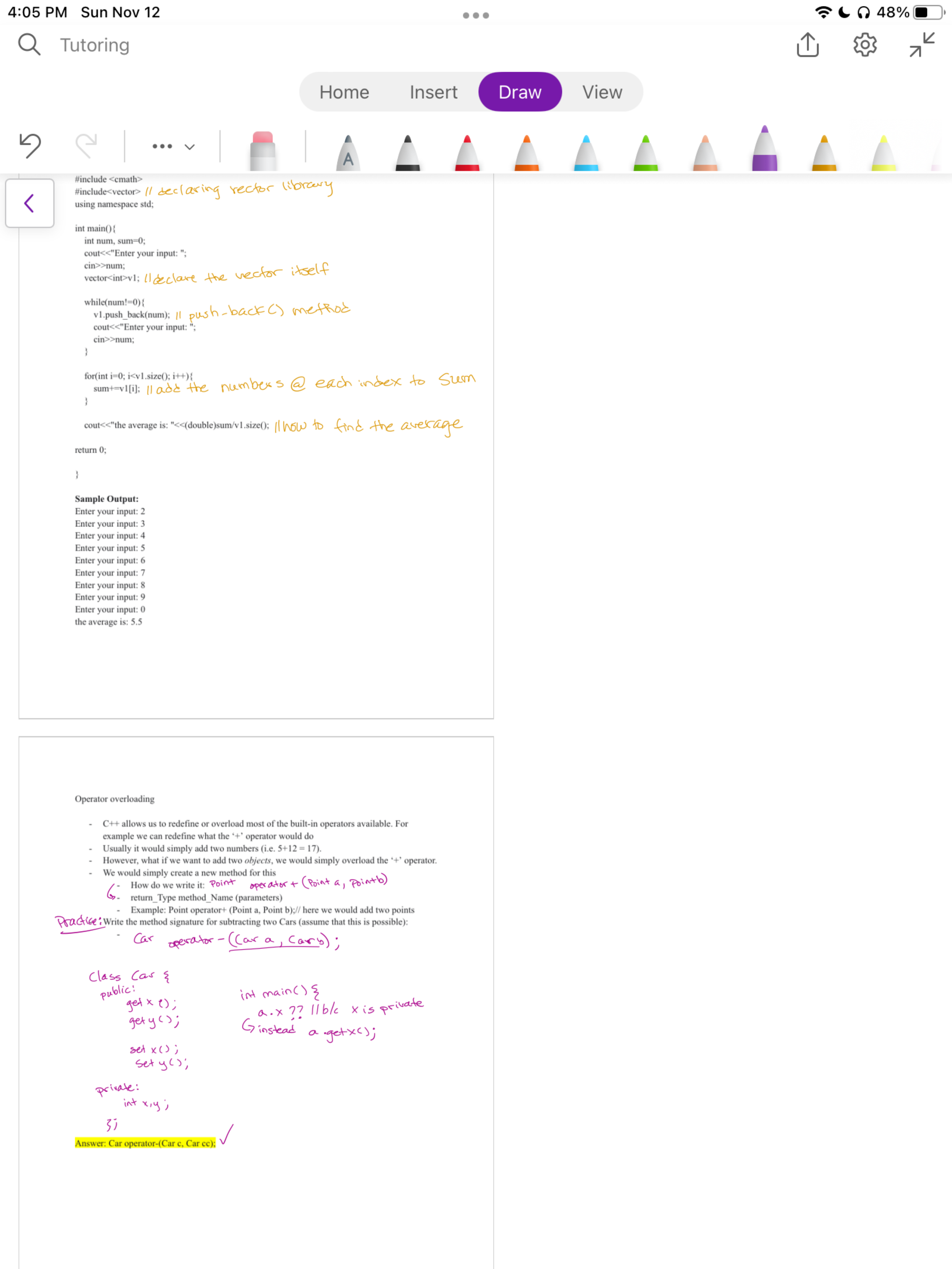
cout << 2[a]; // 2
The compiler:
(2+a)*

- Declare a vector of Cars with size 59. (name wtv you want)
  - vector <Cars> f6 (56).

Full programs using vectors:
- 1) Ask the user to enter some numbers, store those inputs in a vector, and then find the average of all the numbers.

**if we have time we will come back to this, but the answer is on the next page and this will be posted on blackboard.

```
#include <iostream>
#include <cmath>
#include<vector> // declaring vector library
using namespace std;
```



```
#include <cmath>
#include<vector> // declaring vector library
using namespace std;

int main(){
   int num, sum=0;
   cout<<"Enter your input: ";
   cin>>num;
   vector<int>v1; //declare the vector itself

   while(num!=0){
      v1.push_back(num); // push_back() method
      cout<<"Enter your input: ";
      cin>>num;
   }

   for(int i=0; i<v1.size(); i++){
      sum+=v1[i]; // add the numbers @ each index to sum
   }

   cout<<"the average is: "<<(double)sum/v1.size(); //how to find the average

return 0;

}
```

Sample Output:

```
Enter your input: 2
Enter your input: 3
Enter your input: 4
Enter your input: 5
Enter your input: 6
Enter your input: 7
Enter your input: 8
Enter your input: 9
Enter your input: 0
the average is: 5.5
```

Operator overloading

- C++ allows us to redefine or overload most of the built-in operators available. For example we can redefine what the '+' operator would do
- Usually it would simply add two numbers (i.e. 5+12 = 17).
- However, what if we want to add two *objects*, we would simply overload the '+' operator.
- We would simply create a new method for this
  - How do we write it: Point operator+ (Point a, Point b)
  - return_Type method_Name (parameters)
  - Example: Point operator+ (Point a, Point b);// here we would add two points

Practice: Write the method signature for subtracting two Cars (assume that this is possible):

- Car operator-((Car a, Car b));

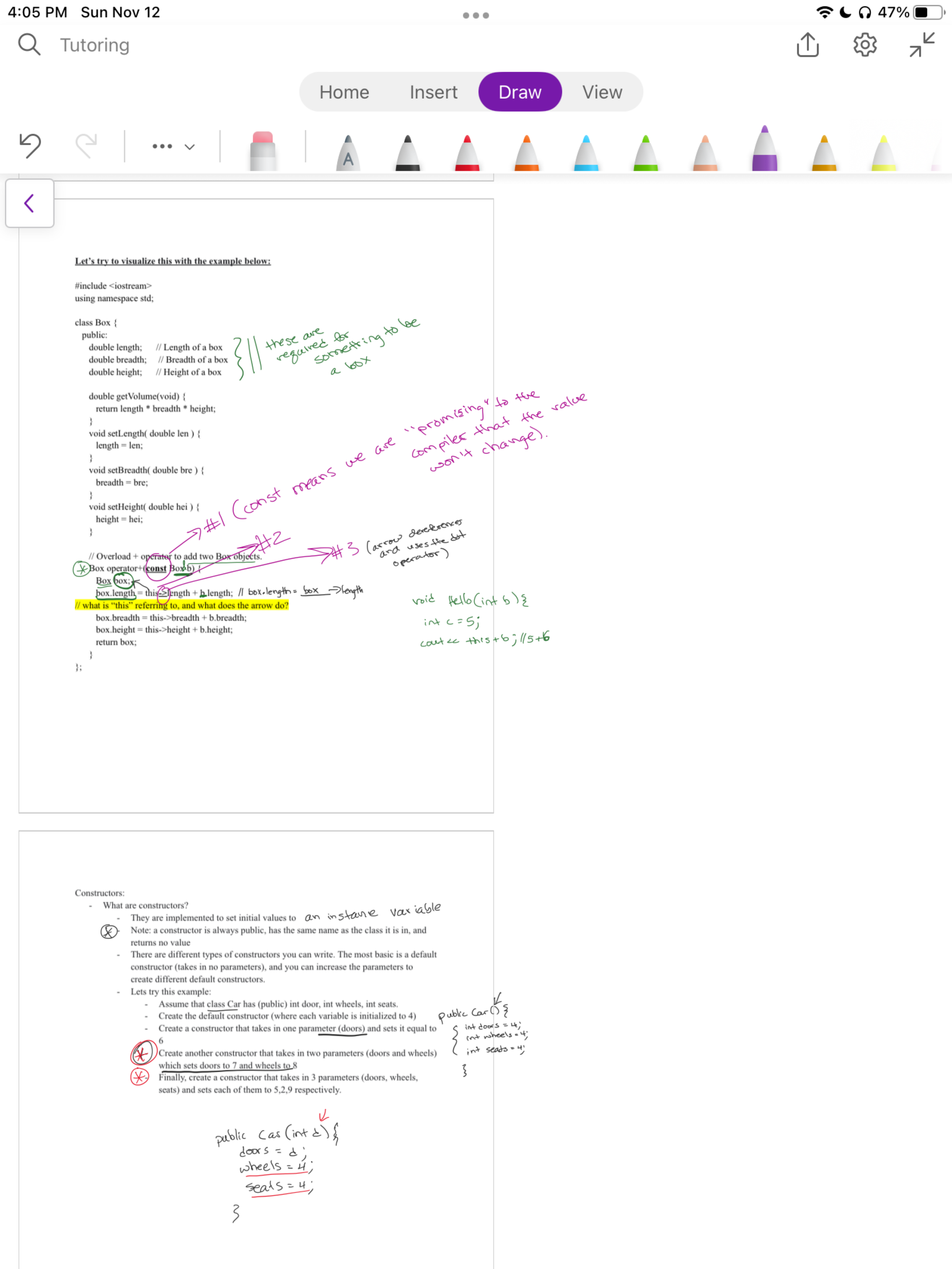
```
Class Car {
 public:
      get x ();
      get y();

      set x();
      set y();

 private:
      int x,y;
 };
```

```
int main(){
   a.x ?? //b/c x is private
 instead a.getx();
```

Answer: Car operator-(Car c, Car cc); ✓



Let's try to visualize this with the example below:

```
#include <iostream>
using namespace std;

class Box {
  public:
    double length;     // Length of a box
    double breadth;    // Breadth of a box
    double height;     // Height of a box

    double getVolume(void) {
      return length * breadth * height;
    }
    void setLength( double len ) {
      length = len;
    }
    void setBreadth( double bre ) {
      breadth = bre;
    }
    void setHeight( double hei ) {
      height = hei;
    }

    // Overload + operator to add two Box objects.
(*) Box operator+(const Box& b) {
      Box box;
      box.length = this->length + b.length;  // box.length = box ->length
// what is "this" referring to, and what does the arrow do?
      box.breadth = this->breadth + b.breadth;
      box.height = this->height + b.height;
      return box;
    }
};
```

}|| these are required for something to be a box

#1 (const means we are "promising" to the compiler that the value won't change).

#2

#3 (arrow dereferences and uses the dot operator)

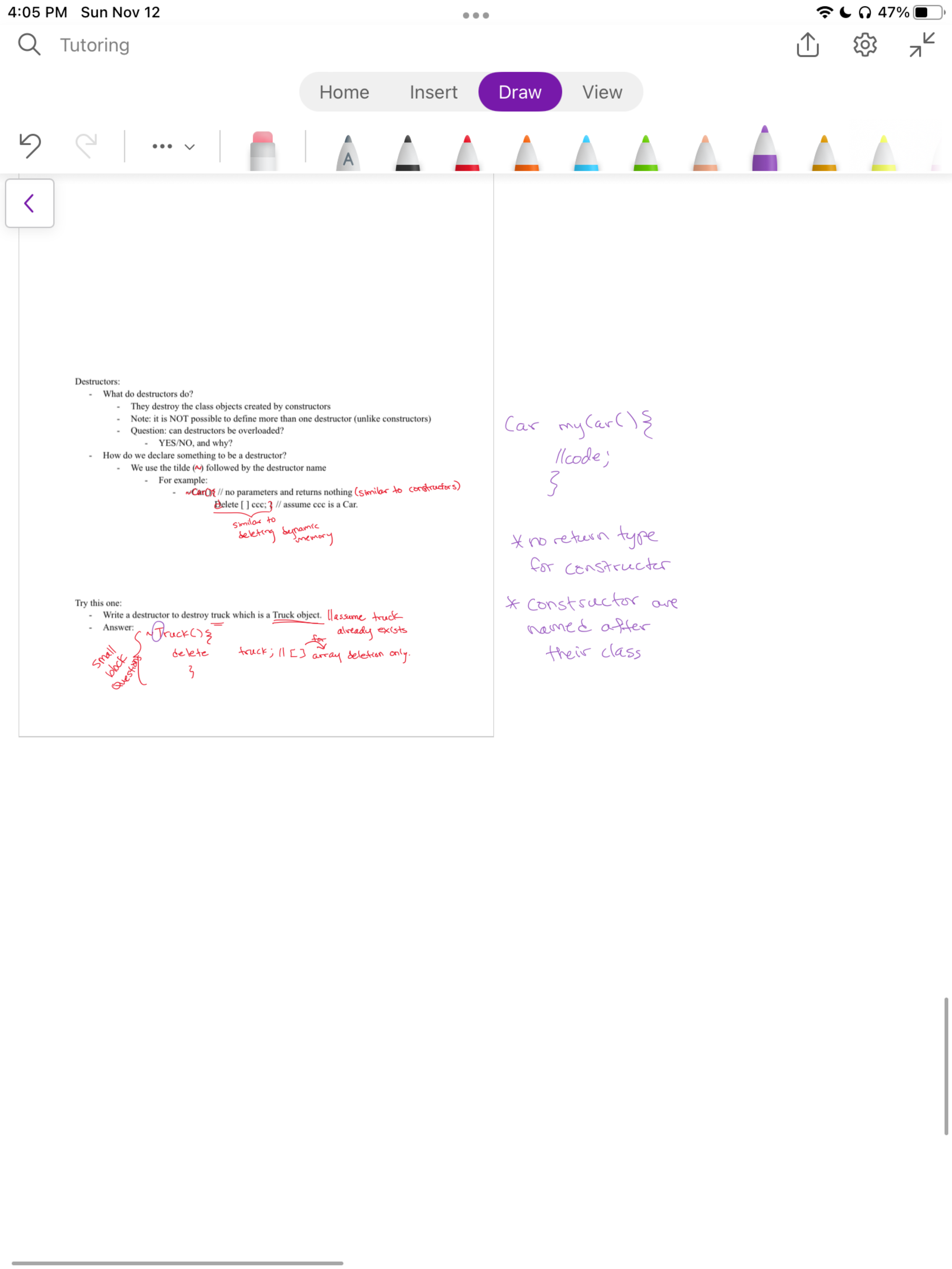
```
void Hello(int 6){
  int c = 5;
  cout << this+6; //5+6
```

Constructors:

- What are constructors?
  - They are implemented to set initial values to an instane variable
  - (*) Note: a constructor is always public, has the same name as the class it is in, and returns no value
  - There are different types of constructors you can write. The most basic is a default constructor (takes in no parameters), and you can increase the parameters to create different default constructors.
  - Lets try this example:
    - Assume that class Car has (public) int door, int wheels, int seats.
    - Create the default constructor (where each variable is initialized to 4)
    - Create a constructor that takes in one parameter (doors) and sets it equal to 6
    - (*) Create another constructor that takes in two parameters (doors and wheels) which sets doors to 7 and wheels to 8
    - (*) Finally, create a constructor that takes in 3 parameters (doors, wheels, seats) and sets each of them to 5,2,9 respectively.

```
public Car(){
  int doors = 4;
  int wheels = 4;
  int seats = 4;
}
```

```
public Cas(int d){
  doors = d;
  wheels = 4;
  seats = 4;
}
```



Destructors:

- What do destructors do?
    - They destroy the class objects created by constructors
    - Note: it is NOT possible to define more than one destructor (unlike constructors)
    - Question: can destructors be overloaded?
        - YES/NO, and why?
- How do we declare something to be a destructor?
    - We use the tilde (~) followed by the destructor name
        - For example:
            - ~Car(){ // no parameters and returns nothing (similar to constructors)

              delete [ ] ccc; } // assume ccc is a Car.

              similar to deleting dynamic memory

Try this one:

- Write a destructor to destroy truck which is a Truck object. //assume truck already exists
- Answer:

```
~Truck(){
    delete        truck; // [ ] for array deletion only.
}
```

small block questions

```
Car myCar(){
    //code;
}
```

* no return type for constructor

* constsuctor are named after their class