1) What kind of classes or functions allow us to specific a type to use? Note: the type is not determined until we call the function.

i.e., we have a function which accepts 2 parameters, both types are what we specified.

ANSWER: Template. A template will allow us to use multiple types using the same code. myFunc<int>(1, 3). myFunc<Compex>(c1, c2). We both use myFunc, but specify the type. This is *not* function overload. We are specifying a type, which makes this a template function.

2) Look at the following code. Then, tell me (1) what makes this possible and (2) the type of z and c.

Segment 1:	Segment 2:
int x = 0;	Complex a = 0;
int y = 100;	Complex b = 24;
auto z = x + y;	auto c = a + b;

WHAT MAKES THIS POSSIBLE? Function overloading. Specifically, operator overloading. We overload `operator+`
WHAT IS THE TYPE OF Z? int. We know that int + int = int**.**
WHAT IS THE TYPE OF C? Technically, we do not know. If we are good programmers, then it should be Complex**. i.e.,** Complex + Complex = Complex**. But if we are bad programmers, then we could do** Complex + Complex = double **or something similar *(do not do this! It truly is horrible!)*.**

3) What are public, private, protected, internal, file-private considered?

ANSWER: These are access modifiers. They declare who can see what within a class. Public – everybody can see. Protected – only those who inherit and itself can see this. Private – only itself can see this (Default in C++). The next two are not directly related to

C++ but are still important to know. **Internal – Restricts to being visible in the nearest module. File-private – can only be seen from within the file only.**

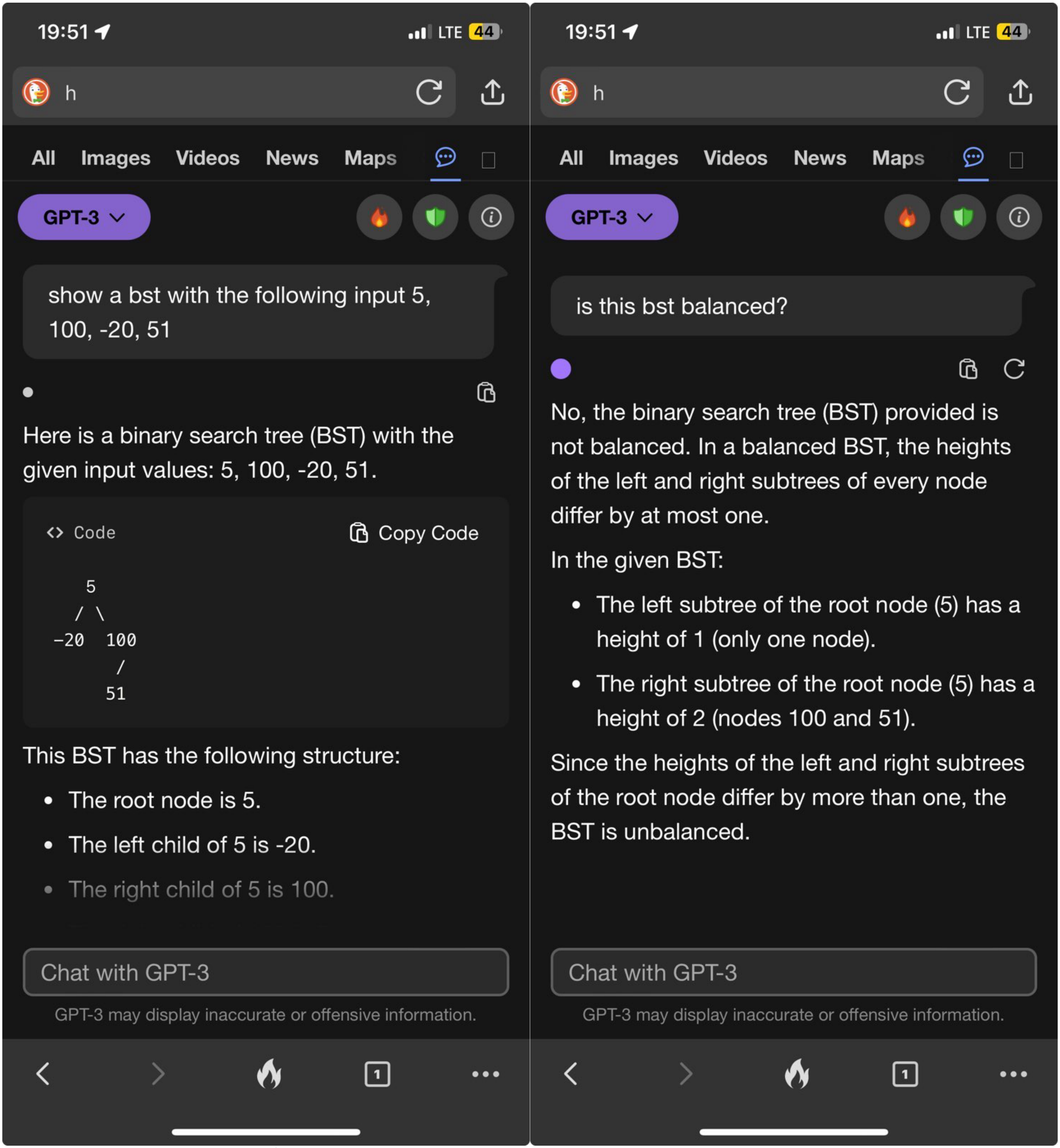
4) What is the difference between the statements below? Be specific, also include any functions that are called, explicitly or implicitly.

a) Complex& operator-(Complex lhs, Complex rhs)
b) Complex& operator-(Complex& lhs, Complex rhs)
c) Complex& operator-(Complex lhs, Complex &rhs)
d) Complex operator-(Complex &lhs, Complex rhs)

ANSWER: For the parameters without `&`, the copy constructor is implicitly called. `&` means pass by reference, so a reference to lhs/rhs was passed which means that it was not copied and we are directly referencing the object.
For the return statements, & passes the return statement by reference. This means we would need to modify lhs or rhs. If we return a local variable by reference, it is like returning an address on the local stack. It will crash when we use it later.
With the previous paragraph, we can conclude that (a) is invalid since we are returning local stack memory. It also makes (b) and (c) non-ideal since they are modifying either lhs or rhs (only valid candidate for return).
Ideally, we would return Complex**, and have lhs and rhs be of type `**const Complex &**` to ensure they are not modified and a copy (the copy constructor is implicitly called here) is returned to for use. There are many ways to do this. You may have also elaborated on other good key points. That would be great!**

5) Suppose we have a file containing 5, 100, -20, 51. We insert these into a BST. Is this tree balanced? Draw the diagram.

ANSWER: Partially. Tree is correct. It is balanced, so it incorrectly identifies the tree as unbalanced.



show a bst with the following input 5, 100, -20, 51

Here is a binary search tree (BST) with the given input values: 5, 100, -20, 51.

<> Code Copy Code

```
    5
   / \
 -20  100
       /
      51
```

This BST has the following structure:

- The root node is 5.
- The left child of 5 is -20.
- The right child of 5 is 100.

is this bst balanced?

No, the binary search tree (BST) provided is not balanced. In a balanced BST, the heights of the left and right subtrees of every node differ by at most one.

In the given BST:

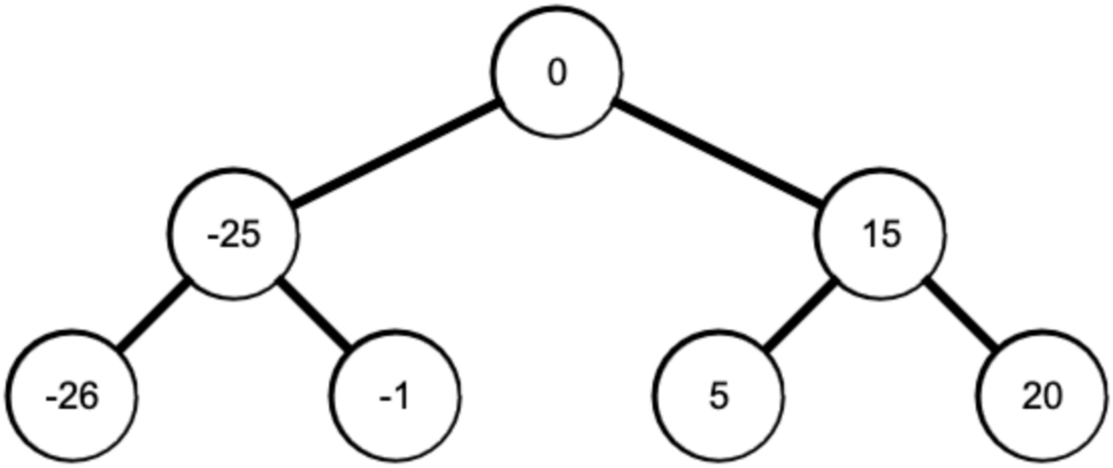
- The left subtree of the root node (5) has a height of 1 (only one node).
- The right subtree of the root node (5) has a height of 2 (nodes 100 and 51).

Since the heights of the left and right subtrees of the root node differ by more than one, the BST is unbalanced.

Screenshots from DDG GPT-3 Chat Via DDG App

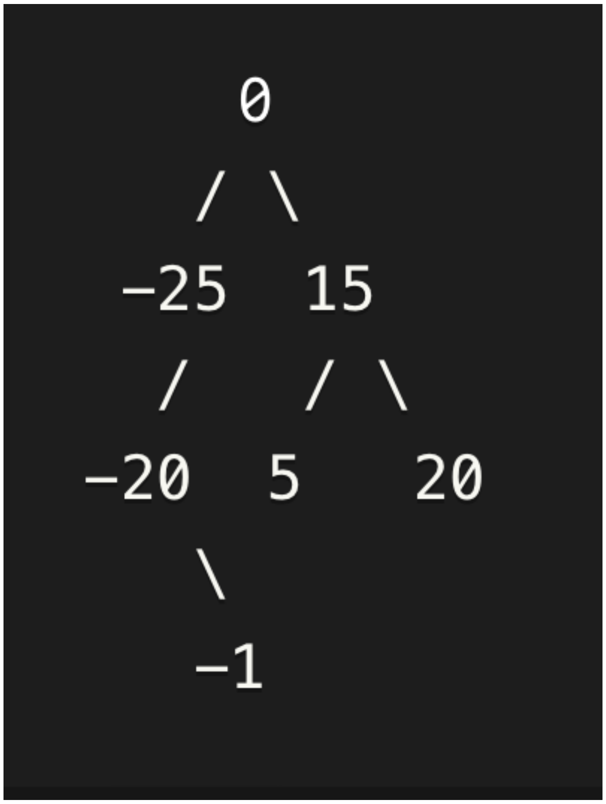
6) Now, suppose we have a file containing 0 15 -25 5 20 -1 -26. Is it balanced? Draw a diagram if it helps.

ANSWER: Yes, this is perfectly balanced.



Src: https://wintercore.github.io/bst-visualizer/

IMPORTANT NOTE: GPT-3 WAS INCORRECT! See the following as an example of why we *need* to double check AI.

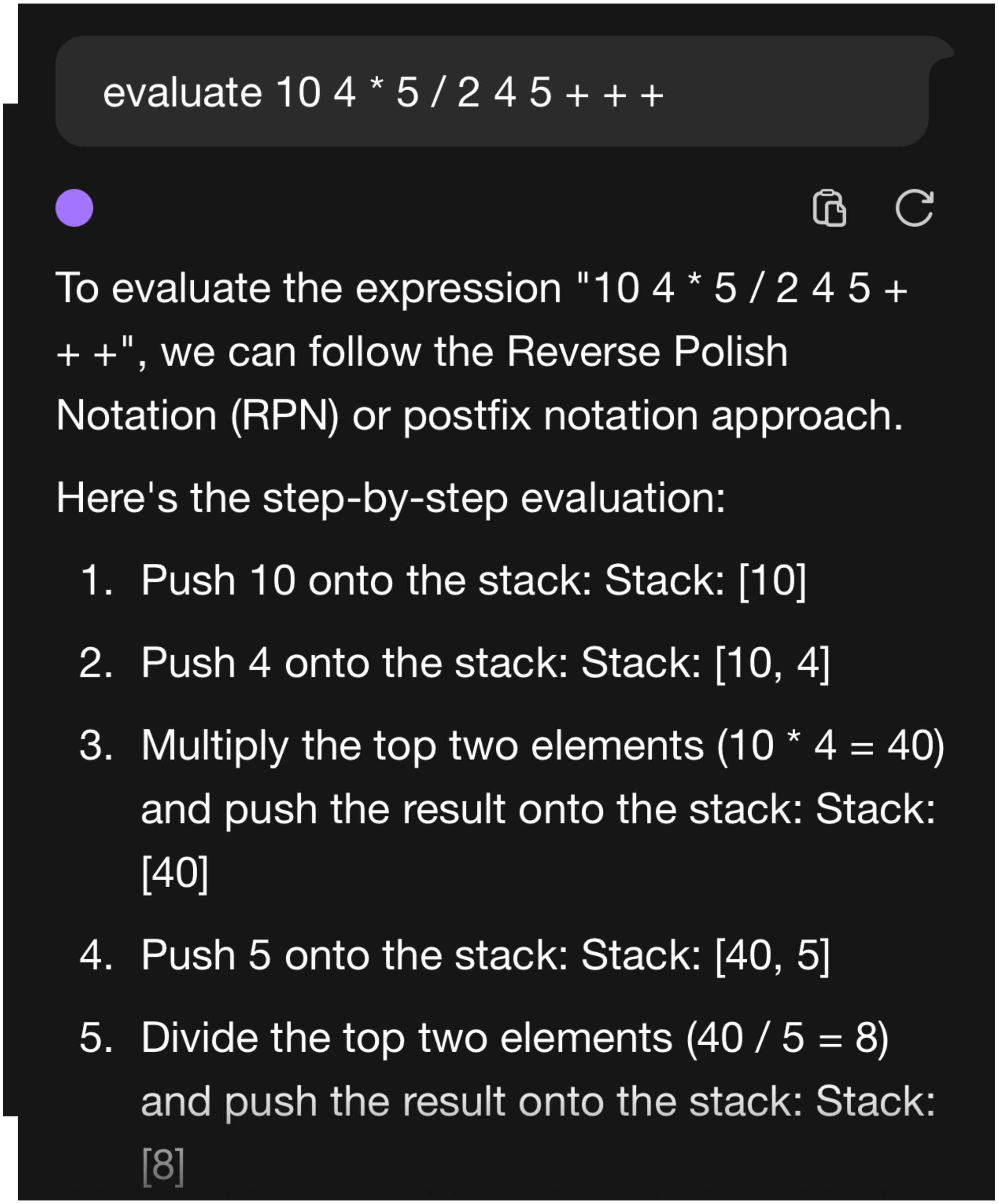


```
    0
   / \
 -25  15
  /   / \
-20  5   20
   \
   -1
```

Screenshots from DDG GPT-3 Chat Via DDG App

7) Evaluate the following computation.
10 4 * 5 / 2 4 5 + + +

ANSWER: 19. See below for how to solve it.



evaluate 10 4 * 5 / 2 4 5 + + +

To evaluate the expression "10 4 * 5 / 2 4 5 + + +", we can follow the Reverse Polish Notation (RPN) or postfix notation approach.

Here's the step-by-step evaluation:

1. Push 10 onto the stack: Stack: [10]
2. Push 4 onto the stack: Stack: [10, 4]
3. Multiply the top two elements (10 * 4 = 40) and push the result onto the stack: Stack: [40]
4. Push 5 onto the stack: Stack: [40, 5]
5. Divide the top two elements (40 / 5 = 8) and push the result onto the stack: Stack: [8]

Screenshots from DDG GPT-3 Chat Via DDG App

Know about:

1. **Information hiding**
2. **Typedef**
3. **Template**
4. **Function overloading**
5. **Copy constructors (when they are called)**
6. **Constructors**
7. **Destructors**
8. **How to pass streams**
9. **Abstraction**
10. **\`this\`**
11. **Stacks**
12. **Queues**
13. **BSTs**

Be able to:

1. **Write a copy constructor given a class**
2. **Write an output stream insertion operator**
3. **Write an input stream operator for an object**
    1. **i.e., cin >> Complex; // You need to write the overloaded op**
4. **Write both a member and non-member binary overloaded operator function**
    1. **i.e., CLASS & CLASS::operator= (CLASS & rhs) // this is lhs, implicitly**
    2. **i.e. CLASS & operator= (CLASS & lhs, CLASS & rhs)**