Announcement

- Homework 7 is released! Due: April 15th.
- Introduction to final project: April 16th.
- Team of three. Register your team on Piazza.

Today's plan

- → Review Ex 11-2
- → Recap questions
- → In-class Ex 11-3

Ex 11-2: Writing a template class

- Use the template syntax: template< typename T >
- This T is like a parameter that you can use in a given scope.
- e.g. template< typename T > class A {...}; , T can be used within the class scope.
- e.g. template< typename T> func(){...}, T can be used within the function scope.

Ex 11-2: defining template class member functions

```
template< typename T>
class A {
void func() { // defined inside the class scope }
}
```

or

```
template< typename T>
void A<T>::func() {
// defined in another scope
}
```

Ex 11-2: **friend** function in a template class

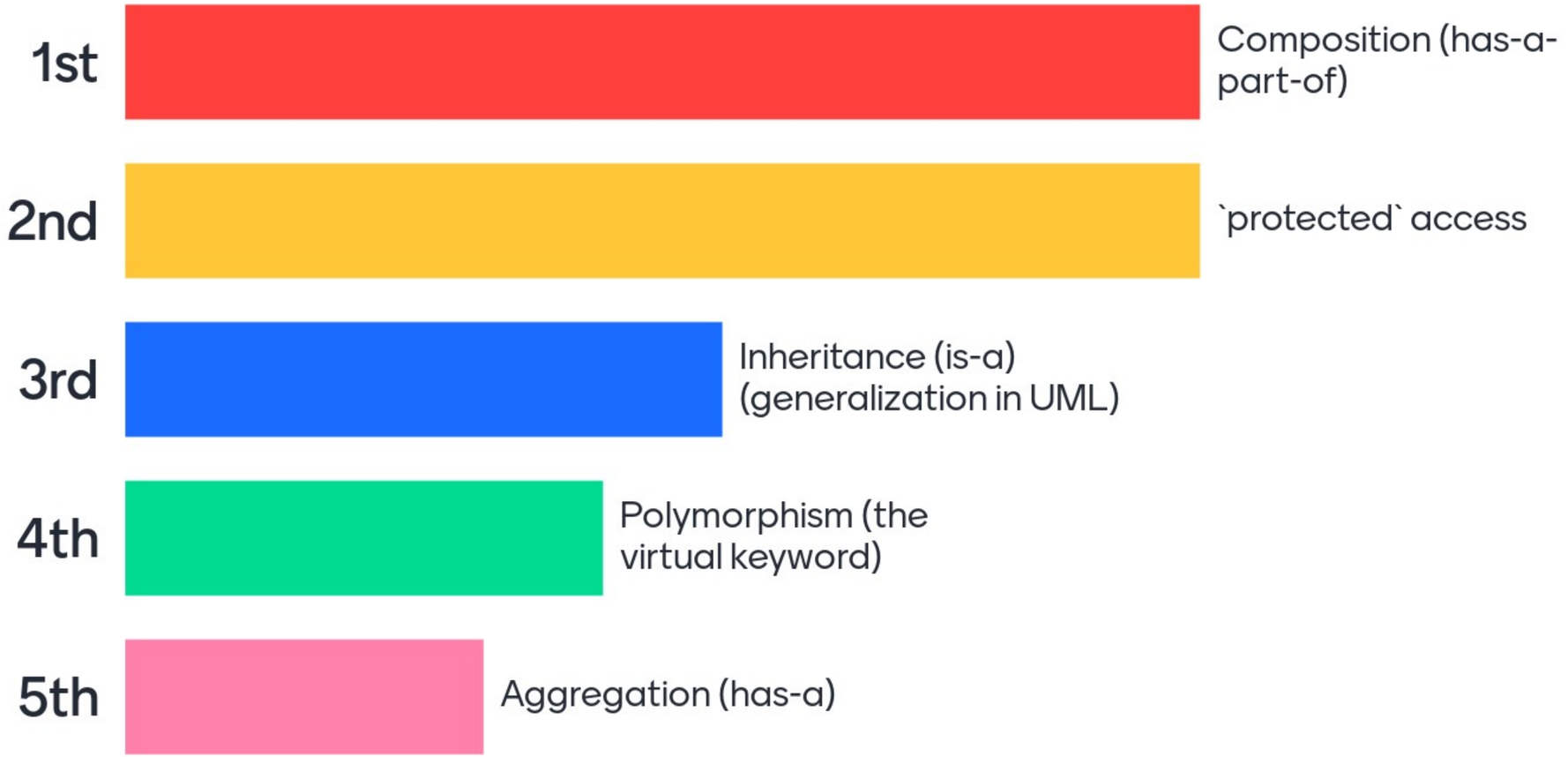
- Recall friend functions are not class member functions
- They don't belong to the class scope
- So, the template parameter T of class A is not seen
- We need another tempalate parameter to declare it is a template function

Ex 11-2: template parameter names

- Same as function parameters, the names should be unique in the scope and meaningful
- We don't need to have the same name in the declaration and the definition
- e.g. template< typename T > class A { void func(); };
- template< typename U > void A<U>::func() {...}

Which one is the most difficult one comparatively?

Mentimeter



2

Different relationships

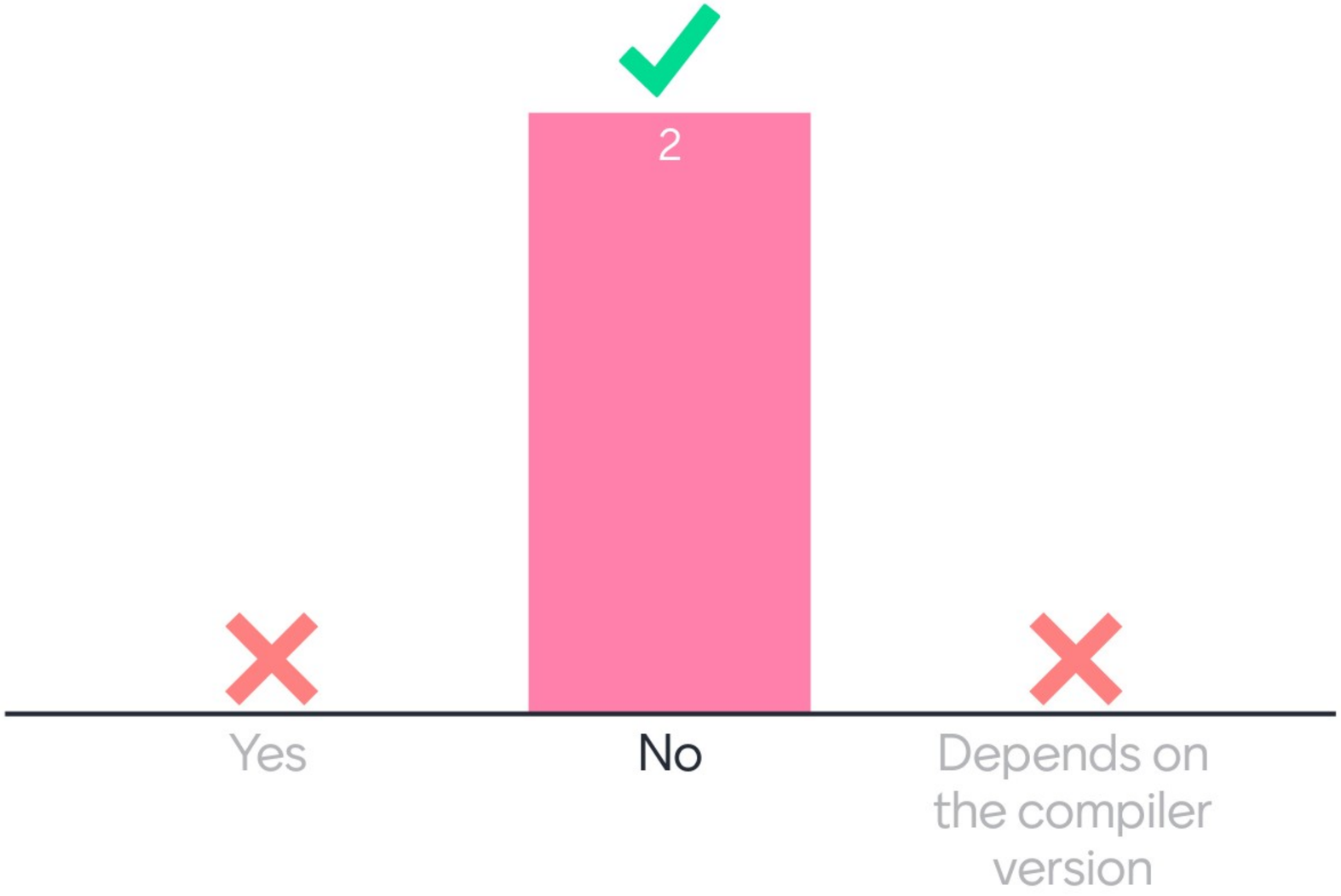
- Inheritance/generalization: An undergraduate is a student
- Realization: A student is a human (implement the abstract class. A special type of inheritance)
- Aggregation: A class has students (when the class is done, the students are still there)
- Composition: A school has classrooms (if the school is destroyed, classrooms also. A special type of aggregation.)

Polymorphism

- Inheritance and abstraction
- A way to use the abstracted interface to develop logic that applies to different realization/implementation
- One abstracted object having multiple concrete implementations is the concept of polymorphsim
- e.g. `BankAccount::withdraw()`: an abstraction of the withdraw action
- By inheritance, we give different concrete implementations for this withdraw action for different account types
- When we develop logic using `BankAccount`, we don't need to know which implementations to use. Polymorphism will help us pick the right one
- More details come next week on how it works in C++ (realized by virtual functions)

Do derived classes inherit constructors?

Mentimeter



2

What does **protected** imply for a class field?

Mentimeter

Private to outside, but accessible from derived classes ✕

The correct answer is: It restricts the access to the class and its derived classes.

1

What is polymorphism?

Mentimeter

When an object has multiple forms ✕

allows for logic specific to each derived class to be implemented ✕

The correct answer is: When an "abstract" (virtual) function is called, but the "concrete" (

2

What is the purpose of the **virtual** keyword?

Mentimeter

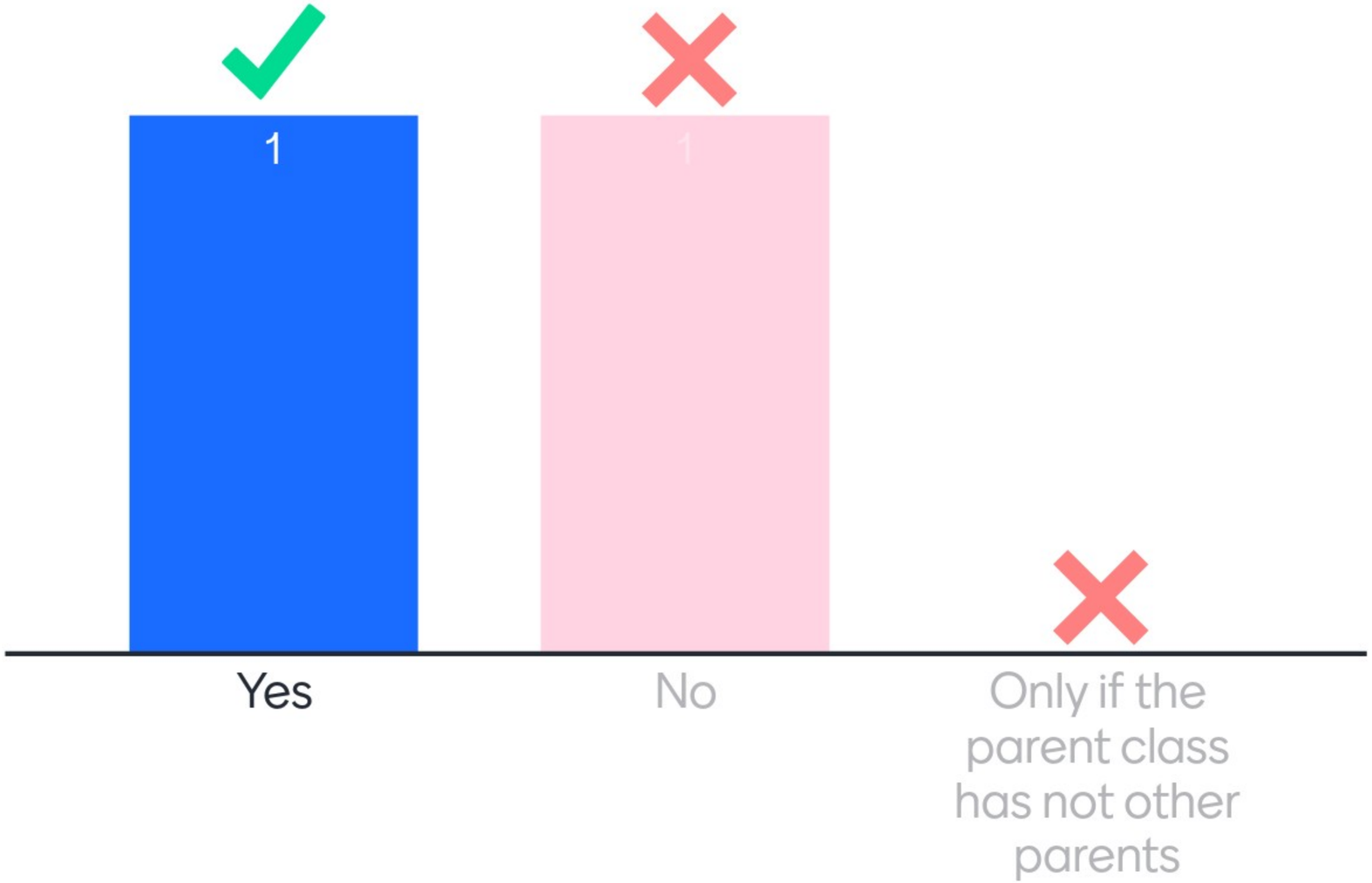
to override a function within a derived class. ✕

The correct answer is: To allow `polymorphism`, i.e. allow the derived class provides an implementation of the abstraction (the virtual function).

1

Can a child class have multiple parent classes?

Mentimeter



2

Ask me anything

Mentimeter

0 questions

0 upvotes