**Multiple inheritance**

- What it is and how it works
- Some languages support it and some don't
- collision

**Multiple inheritance** is a feature in object-oriented programming that allows a class to inherit attributes and methods from more than one parent class. This capability is available in some programming languages like C++ but is not supported in others, such as Java.

**Collision**: If both base classes have a method with the same name, the derived class must specify which method to call using the scope resolution operator.

**Note:** Languages like Java do not support multiple inheritance directly to avoid such complexities.

```
class A {
        public:
            void display() { cout << "Class A" << endl; }
};

class B {
        public:
            void display() { cout << "Class B" << endl; }
};

class C : public A, public B {
    // Class C inherits from both A and B
};

void show() {
    C obj;
    obj.A::display(); // Calls display() from class A
    obj.B::display(); // Calls display() from class B
}
```

**Nested if statements and which one the else statement refers to**

- in java else refers to the closest if statement
- forget about indentation

**Definition:** In nested if statements, the else statement is associated with the nearest preceding if statement within the same block.

**Java** ⇒ public attributes are inherited while private attributes are not
**for C++** you are allowed to modify the inherited attributes visibility as public or private

```
class Base {
        protected:
            int value;
```

```
    public:
       int publicValue;
    };

    class Derived : private Base {
    public:
       void modify() {
          value = 10; // Allowed because it's protected in Base
          publicValue = 20; // Allowed even though it's public in Base
       }
};
```

**question from notes and use c++ as an example**

- use C++:
    - Scope resolution
    - if you have a private attribute which is private, in c++ you can use scope resolution to make it visible

**Definition:** The scope resolution operator **::** is used to access hidden variables or functions.

```
class Example {
    private:
       int value;
    public:
       Example(int v) : value(v) {}
       void display();
    };

    void Example::display() {
       cout << "Value: " << value << endl;
    }
}
```

**Note:** You can use it to access a global variable if there's a local variable with the same name.

**Ternary Operator**

- **Definition:** A conditional operator that assigns a value based on the evaluation of a boolean expression.

```
int count = 5;
int x = (count == 0) ? 0 : 7; // x will be 7 because count is not 0
```

- count = 5, what is x?
- 7 if the statement fails

**question about side effects of functions**

- **Definition:** A side effect occurs when a function modifies some state or interacts with outside variables in addition to returning a value.

b = a + f(a)

**Is this code compact?**
The code `b = a + f(a)` isn't exactly compact, as the function `f(a)` introduces a side effect (F updates the value of a)
- **Unpredictability:** If `f(a)` changes `a`, the final value of `b` can be unclear.
- **Complexity:** It's harder to trace and debug the change in `a` and its effect on `b`.

**Example without Side-Effects:**
int temp = f(a); // Store the result of f(a) in a temporary variable
b = a + temp; // Now use the temporary variable

**Converting objects**
- narrowing or widening conversions
- specific to generic
- generic to specific

  **Example: Java**
    - Java allows widening conversions **implicitly.**

  int myInt = 9;
  double myDouble = myInt; // Implicit widening conversion

    - Java requires explicit casting for narrowing conversions.

  double myDouble = 9.78;
  int myInt = (int) myDouble; // Explicit narrowing conversion

    - conversions really depend on the language

**questions on short circuit evaluations**
- **Definition:** Logical operations can short circuit to avoid unnecessary calculations.
- (x+7)7(b+2)*2
- can happen in either arithmetic or logical calculation
- compiler can evaluate left to right
- if x+7=0, then everything is 0
- compilers are trying to be more efficient
- x = -7 will result in short circuit evaluation if the language evaluates from left to right
- if p = x and y
- if x is false, y value is irrelevant and won't be invoked to calculate
- if p = x or y
- if x is true, y value is irrelevant

**questions on polymorphism**

- **Definition:** Polymorphism allows objects of different classes to be treated as objects of a common superclass.
- When the behaviour of the object depends on its type

### `typeof` Operator

**Definition:** The `typeof` operator is used to determine the type of a variable or expression at runtime. It returns a string indicating the type of the unevaluated operand.

### `kindof` Operator

**Definition:** The `kindof` operator is used to check if an object is a kind of a specific class, meaning it is an instance of that class or any class that inherits from it.

**understand one of the earlier languages talked about in class**

- fragment of one language and **we have to develop a bnf grammar**

**understand the different types in languages**

- explicit type declaration
- implicit type declaration

## Explicit Type Declaration

**Definition:** The programmer **specifies the data type of a variable when it is declared.** This is common in statically-typed languages to ensure type safety and clarity.

**Definition:** The **compiler infers the data type of a variable** from the context in which it is used. This allows for more concise code and is common in dynamically-typed languages or when using type inference in statically-typed languages.

one part is short answer
one part is development
develop code/answer

4-5 production rules
4-5 questions of prolog and ocaml
tracing or specification and you have to write code for the language
directly similar from the labs

specific questions and syntax errors are not penalized

no static and dynamic scoping

OCaml

```
(* place numbers 1 to 4 in the cell, then check if valid *)
let rec test_numbers sudoku x y =
  let rec test_numbers_inner sudoku x y value =
    if (value = 5) then false else
    let temp_sudoku = (Array.copy sudoku) in
    (temp_sudoku.(x).(y) <- value;
    if (is_valid temp_sudoku x y value) then
      if (check_initial_grid temp_sudoku 0 0) then true else (test_numbers_inner sudoku x y
(value+1))
    else (test_numbers_inner sudoku x y (value+1))) in
  (test_numbers_inner sudoku x y 1);;
```

PL

```
right_angle(Curr, Base) :-
    Curr =< Base,
    print_hash(Curr),
    nl, % new line
    NewCurr is Curr + 1,
    right_angle(NewCurr, Base).
right_angle(_, _). % Curr <= Base,
```