CPP-04

★ Intuition lvalues, rvalues

⇒ Every expression is an lvalue or an rvalue.

⇒ lvalues can be written on the left of assignment operator (=)

⇒ rvalues are all other expressions.

⇒ Explicit rvalue defined using &&

⇒ Use std::move (...) to explicitly convert an lvalue to an rvalue.

⇒ The value after move is undefined.

⇒ Moving a variable transfer ownership of ~~a~~ its resources to another variable.

★ Custom operators for a class

⇒ Operators are functions with a signature:

<RETURN_TYPE> operator<NAME> (<PARAMS>)

⇒ <NAME> represents the target operation, e.g. >, <, =, ==, << etc.

★ Copy Constructor

⇒ Called automatically when object is copied.

⇒ For a class MyClass has the signature:

MyClass (const MyClass& other)

* Copy assignment operator

↳ Called automatically when the object is assigned a new value from an LValue.

⇒ For class MyClass has a signature:

MyClass& operator= (const MyClass& other)

Example

MyClass a; // Calling default Constructor

MyClass b=a; // Calling Copy constructor

a=b; // Calling copy assignment constructor.

⇒ Use *this form within a function of a class to get a reference to the current object.

* Move constructor

↳ Called automatically when the object is moved.

↳ For a class MyClass has a signature

MyClass (MyClass&& other)

MyClass c = std::move(a) // Move Constructor

* Move assignment operator

⇒ Called automatically when the object is assigned a new value from an RValue

⇒ For a class MyClass has signature:

```
MyClass& operator=(MyClass&& other)

b= std::move(c) //Move assignment operator
```

Rule of thumb for Constructors & operators:

- ↳ None of them defined : all autogenerated
- ↳ Any of them defined : None autogenerated

{Except for the main Constructor}

⇒ Use =default to use default implementation.

★ <u>Inheritance</u>

⇒ Classes can inherit data and functions from other classes.

⇒ There are 3 types of inheritance in C++:

GOOGLE SHEET (1) Public ⟶ Inheritance keeps all access specifiers of the base class.

(2) Protected

(3) Private

```
Class Derived : public Base {

};
```

⇒ Derived still gets its own special functions: Constructors, destructor, assignment operator.

* Function overriding

⇒ A function can be declared virtual

virtual Func(<PARAMS>);

⇒ If a function is virtual in Base class. it can be overridden in Derived class:

Func(<PARAMS>) override;

⇒ Base can force all Derived classes to override a function by making it pure virtual.

virtual Func(<PARAMS>) = 0;

* Abstract classes and interfaces

Abstract classes → Class that has at least one pure virtual function

interfaces → Class that has only pure virtual functions and no data member