**Copy Constructor:** contains the logic for copying a source object to a newly created object of the same type (initializing to an existing object -> allocate memory to copy object -> return object by value) (Student (const Student&) **Shallow Copies:** directly passing the value (usually number) **Allocate memory:** copy required loop or function to pass the value  **Copy Assignment Operator:** contains the logic for copying data from an existing object to an existing object. This operator will get called when anywhere in the code has identifier = identifier. (check self-assignment -> shallow copies -> deallocated memory -> allocated new memory -> copy data to the newly allocated memory) (Student& operator=(const Student&)) **Localization:** either call the copy assignment from copy constructor or create a private member and localize the common code and call that function  **Direct Call:** \*this = that **Stream Object:** by specifying istream/ostream, code implementation can now ease of use to the destination stream (cout, cerr, clog,...) **Inheritance:** a transitive structural relationship (contain) **Terminology:** A Derived Class inherits the entire structure of the Base Class, access private member through protected function. **Shadowing:** use to access constructor/destructor of inherited class, can’t directly call. In other way, use using Base::function() to remove code duplication if identical. A constructor in derived class can receive value and pass to constructor in base class to initialize the value ( Student(nm, ng) : Person(nm)). In event of inheriting the entire class, define the new class and grant the inheritance to the base class ( class Instructor : public Person { public: using Person : Person; }). Destructor must be implement for seperate class even if inherited. **Helper Operator:** place outside of class, support the class by identified their parameter types

**Abstract Classes:** contains or inherits a pure virtual function (function has to match exactly with pure virtual function) **Template:** reduce code duplicate amongst type (different functions, served identical purpose, but each for different type) **Ad-Hoc Polymorphism:** apparent of polymorphism. Coercion allows convertible changes in argument’s type to match with the corresponding function parameter (convert letter A to 65 in ASCII, called promotion). Overloading allows function call with the same identifier but different argument types. **Universal Polymorphism:** imposes no restriction on the admissible types. Inclusion addresses the multiplicity of definitions available for a function call. Parametric addresses differences between argument types in a function call and the parameter types in function’s definition **Copy Constructor Inheritance:** Derived (const Derived& identifier) : Base(identifier) **Copy Assignment Operator:** Base::operator=(identifer) or (Base&) \*this=identifier

**Abstraction (Class/UML)**: store code in structure or function and that structure or function multiple time in our work, avoid duplication. UML is the format of a class, where it consist of name, type and attributes, operators with return type and parameters **Encapsulation**: integration of data and logic within a class implementation **Inheritance**: One class inherits the structure of another class **Polymorphism**: A single interface provides multiple implementation **Preprocessor**: interprets all directives creating a single translation unit for the compiler (insert the contents of all #include header and substitutes all #define) **Compiler**: compiles each translation unit separately and creates a corresponding binary version **Linker**: assembles the various binary units along with the system binaries to create one complete executable binary **Syntactics Errors**: minor error that exist during coding **Semantic Errors**: error that break the rules of coding

os << left << setw(max\_sku\_length) << setfill(' ')



