**Inheritance**

The **child class** inherits properties of the **parent class** (all member variables and functions)

* Must #include "SuperClass.h"

Syntax:

class DerivedClass **: public** BaseClass {

/\* Only list inherited members if we wish to change them,

add the new members we're adding \*/

// assume for now public is a keyword

};

Table

Description automatically generated

Note that if we declare a **function** as **protected**, then instantiate a children object. We still cannot call that function in main() (protected grants access to derived class definition only)

The children inherits the members, but cannot access private ones.

Text

Description automatically generated

**Graphical user interface, text

Description automatically generated**

**Overriding vs Overloading**

* Overloading: same function name, different parameter list
  + void foo (int v);
  + void foo (double d);
  + void foo (int i, int j);
* Overriding: redefine a function in derived class (not deleting the Parent's ones)
  + List::insert (int a);
  + OrderedList::insert (int a);

Note that the List::insert (int a) is still there (just that the scope is different) 🡪

* You can also redefine a member var, say “a”, w/ the same name in derived class
  + There will be two copies of “a” – one in base obj. and one in derived class obj.

Text

Description automatically generated

**Mixing Types (motivation for polymorphism)**

We can treat the child object as the parent object (ignoring all its additional values), but you cannot treat the parent object as a child

🡪 We can always go up (more general) but never go down (more specific)

🡪 This is intuitive, because we child object has more than the parent object

Diagram, text

Description automatically generated with medium confidence

Usage: We can mix different types of child into a similar data structure, by treating them similarly by their parents

Player array[5]; // We can fit in here every children of Parent

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Warrior | Archer | Wizard | Warrior | Sniper |

**However,** this has some limitations. Say that we overrode the attack() function in every child classes (Warrior do melee, Archer do ranged, etc.). Then if we want every Players in the array to attack by doing:

for (int i = 0; i < 5; i++){

array[i].attack(); // This is because we defined the array of Player

}

This will call the unimplemented attack() version of the parent Player class! **LIMITATION!**

Therefore, there developed an idea of **polymorphism!**

* Note that if we do Player\* base\_class\_ptr still has no idea about the newly defined member variables and methods of the derived class.
* Player\* base\_class\_ptr will simply treat the derived class as the base class.

**Notable questions:**

Base a;

Derived b;

a = b; **// This will call Base operator= (Base &rhs) 🡪 b can fit into rhs**

b = a; **// This will call Derived operator= (Derived &rhs) 🡪 a can't fit into rhs**

It is possible to do:

Base \*a = new Derived ( ); // Still creates a Derived object, but through a, it is

// treated as a Base object.