CS 162, Lecture 12: Inheritance Details
Review: Inheritance

- The process by which a new class is created from another class
- Base (Parent) class == more general class which derived class are created from
- Derived (Child) class == new class
  - Has all of the member variables and functions as base class
- class Derived:public Base {};
- In the Derived class:
  - List only member variables you want to add, not what is inherited
  - Only redefine inherited member functions if you want to redefine them
    - Redefining is when an inherited member function definition is changed in the derived class
- Derived classes can be used anywhere the base class would be used but not the other way around
Interface

• Declare the parent as normal
• The child:

class Child:public parent {
    private:
        // any members which are unique to the child
    public:
        Child(); // default constructor
        // other members including redefined functions from parent
};
Inheritance with the Big Three

- Recall: Big 3 are needed whenever there is dynamic memory or pointers, they are not inherited from the parent. To use in child successfully, they must be defined correctly in parent.

```cpp
Child& Child::operator = (const Child& other) {
    Parent::operator = (other); // invoke parent class AOO
    // continue with things unique to child
}
Child::Child(const Child& copy):Parent(copy) {
    // continue with things unique to child
}
Child::~Child() {
    // define as normal, parent will be automatically called after the child completes
    // destructors go in the reverse of constructors calls
}
```
Public vs. Private vs. Protected Inheritance

• Anything public in the parent is public to the child
• Anything private in the parent is private to the child
  • This means the child can not use private parent functions
  • This means the child can not use private member variables by name, have to use the inherited accessor and mutator functions
• Anything protected in the parent is public to the child but private to everyone else
  • This means the child can use protected member variables by name and protected functions by name
Activity: Chess

- Classes:
  - Bishop (move diagonally \(x\) squares)
  - King (move one square any direction)
  - Knight (move in an “L” shape)
  - Pawn (move one square forward or two on the first move)
  - Piece
    - Queen (move like a Bishop and Rook)
    - Rook (move \(x\) squares in the current row or current column)
- Variables: row_pos, col_pos, color, shape, captured, num_moves
- Functions: accessors for each variable, mutators for each variable, constructors for each class, check_valid, move

1. Identify the Parent and Child classes.
2. Identify which variables and functions should be in the Parent and which should be in the Child. For the moment, only use private and public access modifiers.
3. Write the implementation file for the Rook using your designed parent and child classes.
Chess Design

• Parent: Piece
  • row_pos
  • col_pos
  • Color
  • Shape
  • Captured
  • Accessors/mutators:
    • Constructor
    • Move

• Children
  • King -> victory
  • Queen
  • Pawn
  • Rook
  • Knight
  • Bishop
  • All need constructor, all need check_valid
How does the implementation change if we use protected in the parent?

```cpp
class Piece {
    protected:
        int row_pos;
        int col_pos;
    ...
}

check_valid_accessors(int r, int c)
    if get_row_pos() == r || get_col_pos() == c
        return

check_valid_protecc(int r, int c)
    if row_pos == r || col_pos == c
        return True
```
Using Inherited Objects in Functions

```cpp
void test_bishop(Bishop& p, int r, int c) {
    bool res = p.check_valid(r, c);
    if (res)
        p.move(r, c);
    else
        cout << "MOVE IS NOT VALID" << endl;
}

void test_piece(Piece& p, int r, int c) {
    bool res = p.check_valid(r, c);
    if (res)
        p.move(r, c);
    else
        cout << "MOVE IS NOT VALID" << endl;
}

Bishop b;
test_bishop(b, 3, 0);  //Will these two function calls have the same result?
test_piece(b, 3, 0);
```