CS 162, Lecture 18: Introduction to Polymorphism

11 May 2018

Polymorphism

- The ability to associate many meanings to one function name by means of a special mechanism know as a virtual function or late binding
- Virtual functions may be used before it is defined
- Late binding (dynamic binding): techniques of waiting until run time to determine the implementation of a procedure

How to Polymorph

- Have a virtual function in the base class declaration, don't change anything else

```
class Piece {
        protected:
                int r_pos;
                int c_pos;
                int color;
                char shape;
                bool captured;
        public:
                Piece();
                void move(int, int);
                //appropriate accessor and mutators here
                virtual bool check_valid(int r, int c);
};
```

How does it change what occurs here?

```
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);
```

Move is not valid

Move is not valid

p

pawn

int num_moves

get_num_moves

get_num_moves()

Best practices

- If the function is declared virtual in the base class, you do not need to redeclare it in the derived class
  - But you should -> reminds the programmer which functions are virtual when working in a derived class
- Override: when you redefine a function in a derived class to be used in a polymorphic fashion
- Redefined: when you redefine a function in a derived class
- Why not make all functions virtual?
  - Increased storage overhead and slows the program down

How it works in the compiler

- If a class has one or more virtual functions, the compiler creates a virtual function table for that class
- The table has a pointer for each virtual function to point to the appropriate code.
- If the definition did not change from the parent, it points to the parent.
- If the definition changed from the parent, it points to the child.

The Return of Pointers

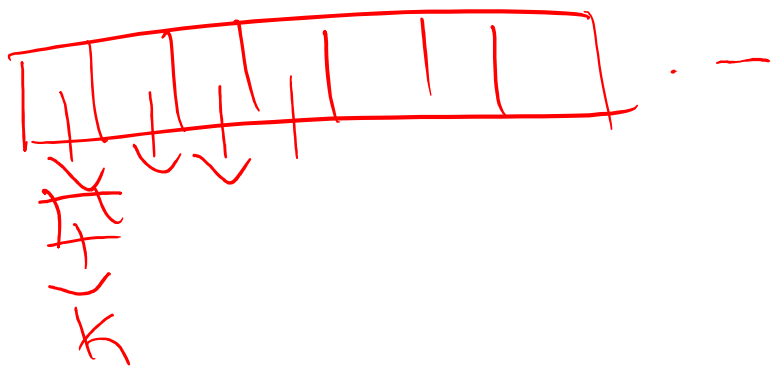
- We know that a base class type can take a child class type (upcasting)
- Slicing problem: lose some features of child class when assigned to the parent
- Fix with pointers, can maintain the child type

```
Piece* p;
Pawn* sacrificial = new Pawn;
p = sacrificial;
p->get_num_moves();
//note: get_num_moves() needs to exist in Piece and must be virtual
```

Exercise

- How would you create a set of chess pieces for a player using polymorphism?

```
Piece** p = new Piece*[16];
p[0] = new King;
⋮
```



Example of Polymorphism Demo

```
Piece** pieces = new Piece*[16];
pieces[0] = new King(color, side);
pieces[1] = new Queen(color, side);
pieces[2] = new Rook(color, side, 0);
pieces[3] = new Rook(color, side, 7);
pieces[4] = new Knight(color, side, 1);
pieces[5] = new Knight(color, side, 6);
pieces[6] = new Bishop(color, side, 2);
pieces[7] = new Bishop(color, side, 5);
for(int i=8; i<16; i++) {
        pieces[i] = new Pawn(color, side, i-8);
}
```

Abstract Classes and Pure Virtual

- Sometimes the function in the base class does not need a definition if the primary purpose is to polymorph into a child
  - Make the function pure virtual in the base class
  - Ex. virtual bool check_valid(int, int) = 0;

Piece* p = new King;

- Abstract class: a class with one or more pure virtual functions.
- Can not have objects of an abstract class which do not polymorph
  - The class does not have complete set of definitions.
- Make destructors virtual, if you don't then the child destructor won't be called

delete [] p