Friendship and Inheritance

OOP in C++

1. Friend Functions

As discussed earlier, private and protected members cannot be accessed from outside the class in which they are declared. However, this does not apply to friends.

Friend functions or classes are declared using the `friend` keyword. A non-member function can access private and protected members of a class iff it is delcared a `friend` of that class. This is done by incluidng a prototype declaration of the external function within the class using the keyword `friend`.

```
#include <iostream>

class Rectangle {
  int width, length;
public:
  Rectangle() {}
  Rectangle(int x, int y) : width(x), length(y) {}
  int area() {
    return width * length;
  }
  friend Rectangle duplicate(const Rectangle&); //Friend function
};

Rectangle duplicate(const Rectangle& rect) {
  Rectangle result;
  //duplicate is able to access width and length since it is declared as a
  ↪  friend of Rectangle.
  result.width = rect.width*2;
  result.length = rect.length*2;
  return result;
}

int main() {
  Rectangle foo;
  Rectangle bar(2, 3);
  foo = dupicate(bar);
  std::cout << foo.area() << std::endl;
  return 0;
}
```

Output:

```
24
```

the `duplicate` function is a friend function of `Rectangle`, therefore it is able to access the private members within `Rectangle`. Notice that `duplicate` is not a member of `Rectangle`. Instead, it is simply just a function that is able to access the private members of `Rectangle`.

2. Friend Classes

Similar to how friend functions are external functions that have access rights to private members, a friend class is a class whose members have acess to the private members of another class.

```
#include <iostream>

class Square;

class Rectangle {
  int width, length;
public:
  int area() {
    return width*length;
  }
  void convert(Square a);
};

class Square {
  friend class Rectangle;
private:
  int side;
public:
  Square(int a) : side(a) {}
};

void Rectangle::convert(Square a) {
  width = a.side;
  height = a.side;
}

int main() {
  Rectangle rect;
  Square sqr(4);
  rect.convert(sqr);
  std::cout << rect.area() << std::endl;
  return 0;
}
```

Output:

```
16
```

In this case, `Rectangle` is a friend class of `Square`. Thus, `Rectangle` is able to access the private member `side` in its methods. Notice that we also created a prototype class declaration `Square`. This is because `convert` takes a `Square` object as an argument, so we must define `Square` first. Note that a friend of a friend is not considered a friend unless explicitly stated.

3. Inheritance

Inheritance involves creating a *derived class* from a *base class*, which "inherits" the members of the base class. For example, imagine that we want to describe different kinds of polygons, including rectangles and triangles. The polygons would share specific members, such as a member function to calculate area. Thus, we can create a base class `CPolygon` that the triangle and rectangle inherit from.

Derived class definition uses the following syntax:

```
class DerivedClassName: public BaseClassName { ... };
```

By default, the members of the base class will be `private` for the derived class. As well, the `protected` access specifier allows derived classes to access members, but not external functions.

```
#include <iostream>

//base class
class Polygon {
protected:
  int width, length; //able to be accessed only through the class Polygon and derived classes
public:
  void setValues(int a, int b) {
    width=a; length=b;
  }
};

class Rectangle: public Polygon {
public:
  int area() {
    return width * length;
    }
};

class Triangle: public Polygon {
public:
  int area() {
    return width * length / 2;
    }
};

int main() {
  Rectangle rect;
  Trangle tri;
  rect.setValues(4, 5); //rect is a Rectangle, which is derived from Polygon. Therefore rect can use member functions defined in Polygon
  tri.setValues(4, 5);
  std::cout << "Rect: " << rect.area() << std::endl;
  std::cout << "Tri: " << tri.area() << std::endl;
  return 0;
}
```

Output:

```
Rect: 20
Tri: 10
```

3.1. What is Inherited from the Base Class?

In principle, a derived class inherits access to every member defined in the base class except for the following:

- Constructors and Destructors*
- Assignment operator overloads
- Friend
- Private members

Even though access to the constructors and destructors is not inherited, they are still called when we create an object of the derived class. As well, the constructors of a derived class calls the default constructor of the base class unless otherwise specified. We can do this with the following syntax:

```
DerivedConstructorName(parameters): BaseConstructorName(parameters) {...}
```

```
#include <iostream>

class Mother {
public:
  Mother() {
    std::cout << "Mother: no arguments" << std::endl;
  }
  Mother(int a) {
    std::cout << "Mother: int argument" << st::endl;
  }
};

class Daughter: public Mother {
public:
  Daughter(int a) {
    std::cout << "Daughter: int argument" << std::endl;
  }
};

class Son: public Mother {
public:
  Son(int a): Mother(a) {
    std::cout << "Son: int argument" << std::endl;
  }
};

int main() {
  Daugher person1(0);
  Son person2(0);

```

```
31   return 0;
32 }
```

Output:

```
Mother: no arguments
Daughter: int argument

Mother: int argument
Son: int argument
```

In the example above, even though we define the constructor for `Daughter` with a parameter, it will still call the default constructor for `Mother`. Only if we explicitly declare to call a different constructor (such as in `Son`) will we not call the default.

3.2. Multiple Inheritance

A class may also inherit from multiple classes. This is done by specifying more base classes, seperated by a comma.

```
1 class Rectangle: public Polygon, public Output
```

```
1  #include <iostream>
2
3  class Polygon {
4  protected:
5    int width, length;
6  public:
7    Polygon(int a, int b): width(a), length(b) {}
8  };
9
10 class Output {
11 public:
12   static void print(int i);
13 };
14
15 void Output::print(int i) {
16   std::cout << i << std::endl;
17 }
18
19 class Rectangle: public Polygon, public Output {
20 public:
21   Rectangle(int a, int b): Polygon(a, b) {} //use Polygon constructor
22   int area() {
23     return width * length; //access protected members
24   }
25 };
26
27 class Triangle: public Polygon, public Output {
28 public:
29   Triangle(int a, init b) Polygon(a, b) {}
30   int area() {
31     return width * length / 2;
```

```
32    }
33  };
34
35  int main() {
36    Rectangle rect(4, 5);
37    Triangle tri = Triangle(4, 5);
38    rect.print(rect.area());
39    Triangle::print(tri.area());
40    return 0;
41  }
```

Output:

```
20
10
```