7. Program Set 5

1. Class example for a point in space

```cpp
#include <iostream>

class Point {
public:
    double x, y;   // public data (like the old struct)
};

int main()
{
    Point p;
    p.x = 1.5;
    p.y = -2.0;
    std::cout << "(" << p.x << ", " << p.y << ")\n";

    return 0;
}
```

2. Nested classes

```cpp
#include <iostream>

class Point {
public:
    double x, y;
};

class Vector {
public:
    Point start, end;
};

int main() {
    Vector v;
    v.start.x = 1.2;
    v.start.y = 0.4;
    v.end.x   = 2.0;
    v.end.y   = 1.6;

    std::cout << "(" << v.start.x << "," << v.start.y << ") -> ("
              << v.end.x   << "," << v.end.y   << ")\n";


    return 0;
}
```

3. Copying one class to another and copying member of one class to another

```cpp
#include <iostream>

class Point {
public:
    double x, y;
};

int main() {
    Point a;

    a.x = 3;
    a.y = 4;
    Point b = a;        // memberwise copy
    b.x = 7;            // independent copy

    std::cout << "a:(" << a.x << "," << a.y << ") b:("
              << b.x << "," << b.y << ")" << std::endl;
    std::cout << "&a: " << &a << ", " << "&b: "
              << &b << std::endl;

    return 0;
}
```

4. Pass class instance by value to function

```cpp
#include <iostream>
class Point {
public:
    double x, y;
};

void offsetPoint(Point p, double dx, double dy) { // copy
    p.x += dx;
    p.y += dy;
}

int main() {
    Point p = {3, 4};

    offsetPoint(p, 1, 2);
    std::cout << "(" << p.x << "," << p.y << ")" << std::endl; //
still (3,4)

    return 0;
}
```

5. Pass class instance by reference to function

```cpp
#include <iostream>

class Point {
public:
    double x, y;
};

void offsetPoint(Point &p, double dx, double dy) { // reference
    p.x += dx;
    p.y += dy;
}

int main() {
    Point p = {3, 4};

    offsetPoint(p, 1, 2);
    std::cout << "(" << p.x << "," << p.y << ")" << std::endl;
                            // now (4,6)

    return 0;
}
```

6. Methods of a class

```cpp
#include <iostream>
class Point {
public:
    double x, y;

    void offset(double dx, double dy) {
        x += dx;
        y += dy;
    }

    void print()
    {
        std::cout << "(" << x << "," << y << ")";
    }
};

int main() {
    Point p;

    p.x = 1.2;
    p.y = 0.4;

    p.offset(1.0, 1.5);
```

```
    p.print();

    std::cout << std::endl;

    return 0;
}
```

7. Calling a sub-objects method

```
#include <iostream>
class Point {
public:
    double x, y;
    void offset(double dx, double dy)
    {
        x += dx;
        y += dy;
    }

    void print()
    {
        std::cout << "(" << x << "," << y << ")";
    }
};

class Vector {
public:
    Point start, end;
    void offset(double dx, double dy)
    {
        start.offset(dx,dy);
        end.offset(dx,dy);
    }
    void print()
    {
        start.print();
        std::cout << " -> ";
        end.print();
    }
};

int main() {
    Vector v;

    v.start.x = 1.2;
    v.start.y = 0.4;
    v.end.x   = 2.0;
    v.end.y   = 1.6;   // (no brace-assignments in C++98)
```

```
    v.offset(1.0, 1.5);
    v.print();
    std::cout << std::endl;

    return 0;
}
```

8. Defining methods outside of the class

```
#include <iostream>

class Point {
public:
    double x, y;
    void offset(double dx, double dy);
    void print();
};

class Vector {
public:
    Point start, end;
    void offset(double dx, double dy);
    void print();
};

void Point::offset(double dx, double dy)
{
    x += dx;
    y += dy;

}
void Point::print()
{
    std::cout << "(" << x << "," << y << ")";

}
void Vector::offset(double dx, double dy)
{
    start.offset(dx,dy);
    end.offset(dx,dy);
}
void Vector::print()
{
    start.print();
    std::cout << " -> ";
    end.print();
}
```

```
int main() {
    Vector v;

    v.start.x = 0;
    v.start.y = 0;
    v.end.x   = 1;
    v.end.y   = 1;

    v.offset(2,3);

    v.print();
    std::cout << std::endl;

    return 0;
}
```

9. Default value constructor

```
#include <iostream>

class Point {
public:
    double x, y;
    Point()
    {
        x = 0.0;
        y = 0.0;
    }
};

int main() {
    Point p; // default constructor runs

    std::cout << p.x << "," << p.y << std::endl;

    return 0;
}
```

10. Constructor with parameters

```
#include <iostream>

class Point {
public:
    double x, y;
    Point(double nx, double ny)
    {
        x = nx;
```

```
        y = ny;
    }
};

int main() {
    Point p(2.0, 3.0);

    std::cout << p.x << "," << p.y << std::endl;

    return 0;
}
```

11. Overloaded constructors

```
#include <iostream>

class Point {
public:
    double x, y;
    Point()
    {
        x = 0.0;
        y = 0.0;
    }
    Point(double nx, double ny)
    {
        x = nx;
        y = ny;
    }
};

int main() {
    Point a;          // default

    Point b(5.0, 6.0); // 2-arg

    std::cout << a.x << "," << a.y << " | " << b.x << "," << b.y <<
std::endl;

    return 0;
}
```

12. Copy constructor

```
#include <iostream>

class Point {
public:
```

```
    double x, y;
    Point(double nx, double ny)
    {
        x = nx;
        y = ny;
    }
};

int main() {
    Point q(1.0, 2.0);

    Point r = q; // copy constructor invoked

    std::cout << r.x << "," << r.y << std::endl;
    std::cout << "&q = " << &q << ": &r = " << &r << std::endl;

    return 0;
}
```

13. Shallow copy issue in copying classes. If any class member is a pointer, and a copy of the class is made, then the data at the memory pointed to by said pointer does not get copied over. Only the address is copied over to the pointer member in the new class instance.

```
#include <iostream>
#include <cstring>

class Student {
public:
    int id;
    char* name;         // owning raw pointer for demo
    Student() : id(0), name(0) {}
};

int main() {
    Student s1;

    s1.id = 42;
    char buf[] = "foo";
    s1.name = buf;

    Student s2 = s1;              // shallow copy: pointers equal
    s2.name[0] = 'b';             // mutates both!

    std::cout << s1.name << std::endl; // prints "boo"

    std::cout << "&s1.id : " << (&s1.id) <<
                 ", &s2.id : " << &s2.id <<
                 std::endl;
```

```
    std::cout << "s1.name : " << static_cast<const void*>(s1.name)
              << ", s2.name : " << static_cast<const void*>(s2.name)
              << std::endl;

    return 0;
}
```

14. Manually create a copy constructor for deep copy

```
#include <iostream>
#include <cstring>

class Student {
public:
    int id;
    char* name;

    Student() : id(0), name(0) {}

    Student(const Student& o) : id(o.id)
    {             // deep copy
        if (o.name)
        {
            name = new char[std::strlen(o.name)+1];
            std::strcpy(name, o.name);
        }
        else
            name = 0;
    }

    Student(int id, const char* str)
    {             // deep copy
        if (str)
        {
            name = new char[std::strlen(str)+1];
            std::strcpy(name, str);
        }
        else
            name = 0;
    }

    ~Student()
    {
        if (name)
            delete[] name; // tidy up
    }
};
```

```
int main() {
    char buf[] = "foo";

    Student s1(7, buf);

    Student s2 = s1;              // deep copy; pointers differ

    if (s2.name)
        s2.name[0] = 'b';

    std::cout << "s1.name: " << s1.name << std::endl; // still "foo"
    std::cout << "s2.name: " << s2.name << std::endl;
}
```

15. Attempt to modify a private class member gives compile error

```
#include <iostream>

class Point {
private:
    double x, y;
public:
    Point(double nx, double ny) : x(nx), y(ny)
    {}
};

int main() {
    Point p(2.0, 3.0);
    p.x = 5.0; // <- Uncomment to see "x is private" error
    std::cout << "ok\n";
}
```

16. getter and setter methods in classes

```
#include <iostream>

class Point {
private:
    double x, y;
public:
    Point(double nx, double ny) : x(nx), y(ny)
    {}
    double getX()
    {
        return x;
    }
    double getY()
    {
```

```
        return y;
    }
};

int main() {
    Point p(2.0, 3.0);

    std::cout << p.getX() << "," << p.getY() << std::endl;

    return 0;
}
```

17. default access behaviour of struct members and class members

```
#include <iostream>

struct S
{
    int a;
};     // public by default

class C
{
    int a;
};      // private by default

int main() {
    S s;

    s.a = 5;          // OK

    C c;
    c.a = 5;       // Uncomment -> error (a is private)

    std::cout << s.a << std::endl;

    return 0;
}
```

18. array of class objects

```
#include <iostream>

class Point {
public:
    double x, y;
    Point() : x(0), y(0)
    {}
```

```
    void print()
    {
        std::cout << "(" << x << "," << y << ") ";
    }
};

int main() {
    Point pts[3];

    for (int i = 0; i < 3; ++i)
    {
        pts[i].x = i;
        pts[i].y = -i;
    }

    for (int i = 0; i < 3; ++i)
        pts[i].print();

    std::cout << std::endl;

    return 0;
}
```