Introduction to C++ Classes

George M. Tzoumas

National University of Athens

Programming techniques

- unstructured programming
- procedural programming
- modular programming
- object-oriented programming

Unstructured programming

Program

main program data

Procedural programming

main program	*procedures*
call proc_1	code of proc_1
call proc_4	code of proc_2
call proc_2	code of proc_3
call proc_3	code of proc_4

Modular Programming

main progarm

data

module_1

data

procedures

module_2

data

procedures

Object-oriented Programming

Object_1

attr_1,attr_2

$\text{method}_1()$
$\text{method}_2()$

Object_2

attr_1

$\text{method}_1()$
$\text{method}_2()$

Example objects

Triangle

$x_1, y_1, x_2, y_2, x_3, y_3$

---

draw()
rotate()

Circle

x, y, r

---

draw()

C++ Class Declaration

```
class Point {
public:
    int x, y;

    Point() {          // constructor
        x = y = 0;
    }

    void translate(int dx, int dy) {
        x += dx;
        y += dy;
    }
};
```

Creating objects of the class

```
int main()
{
    int b;
    Point p;                // p is (0,0)

    p.translate(4,4); // p is (4,4)
    b = p.x * p.y;          // b is 16
}
```

More details...

```
class Point {
private:
    int old_x, old_y;

public:
    int x, y;

    Point() {
        x = y = old_x = old_y = 0;
    }

    Point(const int ax, const int ay) {
        old_x = x = ax; old_y = y = ay;
    }

    void translate(const int dx, const int dy) {
        x += dx; y += dy;
    }

    void revert() {
        x = old_x; y = old_y;
    }
};
```

Class in use

```
int main()
{
    Point q(3,-1);      // q is (3,-1)

    q.translate(2,7); // q is (5,6)
    q.revert();         // q is (3,-1)
}
```

An advanced example

```
class Intfile {
private:
    FILE *fp;

public:
    Intfile(const char *path, const char *mode) {
        fp = fopen(path, mode);
    }

    ~Intfile() {
        if (fp != NULL) fclose(fp);
    }

    int readint() {
        int n;
        fscanf(fp, "%d", &n);
        return n;
    }

    void writeint(int n) {
        fprintf(fp, "%d\n", n);
    }
};
```

Using Intfile class

```
int main()
{
    Intfile r("in.dat", "r");          // constructor of r
    int a = r.readint();
    if (a > 0) {
        Intfile w("out.dat", "w"); // constructor of w
        w.writeint(a);
    }                                  // destructor of w
}                                      // destructor of r
```

Inheritance

```
class Point3d: public Point {
private:
    int old_z;
public:
    int z;

    Point3d(): Point() {
        z = old_z = 0;
    }

    Point3d(const int ax, const int ay, const int az): Point(ax, ay) {
        old_z = z;
    }

    void translate(const int dx, const int dy, const int dz) {
        Point::translate(dx, dy);
        z += dz;
    }

    void revert() {
        Point::revert();
        z = old_z;;
    }
};
```

Using Point3d

```
int main()
{
    Point3d *p;
    p = new Point3d(1,2,3);        // p is (1,2,3)
    p->translate(-1,-2,-3);        // p is (0,0,0)
    p->revert();                   // p is (1,2,3)
    delete p;                      // implicit destructor
}
```

```
int main()
{
    Point *k;
    Point3d *p;
    p = new Point3d(1,2,3);        // p is (1,2,3)
    p->translate(-1,-2,-3);        // p is (0,0,0)
    k = p;
    k->revert();                   // p is (1,2,0) !!!
    delete p;                      // implicit destructor
}
```

Virtual functions

```
class Point {
...
    virtual void revert() {
        x = old_x; y = old_y;
    }
};

class Point3d: public Point {
...
    void revert() {
        Point::revert();
        z = old_z;;
    }
};
```

Polymorphism

```
int main()
{
    Point *k;
    Point3d *p;
    p = new Point3d(1,2,3);   // p is (1,2,3)
    p->translate(-1,-2,-3);   // p is (0,0,0)
    k = p;
    k->revert();              // p is (1,2,3)
    delete p;                 // implicit destructor
}
```

Abstract classes

```
class Spirte {
    int x, y;
...
    Sprite() { ... }
    virtual ~Sprite() { }    // empty virtual destructor
    virtual void draw() = 0;
};

class PlayerSprite: public Sprite {
...
    PlayerSprite(): Sprite() { ... }
    void draw() { ... }
};
```

There is more

- Operator overloading
- Namespaces
- I/O
- Templates
- STL
- CGAL

Examples

```
std::vector<int> v;

v.push_back(1);
std::cerr << v[0];
```