friend functions and classes

- sometimes we want a class to have private fields/methods, yet still allow one or more specific other functions or classes to be able to access them
- from inside the class definition we can specify which functions/classes will be given such access, listing them as “friends” of the class
- sometimes done out of convenience/laziness
- sometimes it's the most sensible way to overload certain operators
- sometimes it's the most sensible way to code two classes that are tightly interrelated

syntax to declare a friend function

- to make a function a friend we simply put the keyword friend in front of it
- pass the object to be accessed as a parameter, usually by reference, using const if we don't want the friend to alter content

```
class example {
   private:
      int x, y, z;
   public:
      ...
      friend void AllAccess(example &e);
};
```

```
// our external function that can still access
// private fields/methods
void AllAccess(example &e)
{
     // can view/alter anything in e
     e.x = 105;
}
```

Example: overloading - for negate

- suppose we want to be able to negate x,y in our circle class using - on a circle object

```
class circle {
   private:
      int x, y, radius;
   public:
      circle(int xv, int yv, int rad) {
          x = xv; y = yv; radius = rad; }
      void print() {
           cout << x << y << radius;
      friend void operator-(circle &c);
};
void operator-(circle &c)
{
    c.x = - c.x;
    c.y = - c.y;
}
```

```
int main()
{
     circle mycirc(3,2,11);

     -mycirc; // negate circle coords

     mycirc.print();
     // displays -3 -2 11
}
```

syntax to declare a friend class

- simply specify the class/name as a friend
- all methods in the friend class can access private content of the declaring class

```
class example {
   private:
      int x, y, z;
   public:
      ...
      friend class SomeOtherClass;
};

// all methods of SomeOtherClass can access x,y,z fields
// of any example objects that are passed to them
```