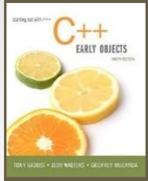
COP2334

Introduction to Object Oriented Programming with C++

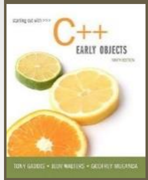
D. Singletary



Module 9

Ch. 11 More about Classes and OOP

Ch. 11 The **this** Pointer

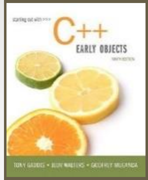


- **this** is an implicit pointer which, when referred to from within an object instance, refers to the object itself
- We can use it to disambiguate when setting a member variable:

```
// mutator to set member variable x
void setX(int x)
{
        // set the member to the value
        // of the parameter passed as x
        this->x = x;
}
```

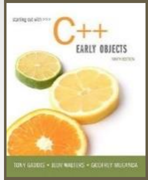
- A programming convention which is frequently followed when passing parameters to constructors or mutators to set a member variable's value is to use the member variable's name as the parameter name; "this" is required in this case in order to follow this convention (we will follow this convention in this course)

Constant Function Parameters



- A parameter passed to a function as a pointer or reference can be modified by the function
- We can use the const keyword when declaring that parameter to prevent modification

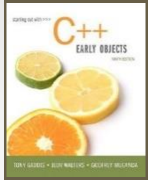
```
void setAValue(const int& value)
{
     // value cannot be modified in this function
     …
}
```



- If we pass a parameter to a function as a constant, we cannot treat it as a non-constant if we pass it to another function:

```
int main()
{
   int x = 25;

   displayValue(x);
   return 0;
}

void displayValue(const int &value)
{
   doubleValue(value); // can't do this, value is const
   cout << "value is " << value << endl;
}

void doubleValue(int &value)
{
   value *= 2;
}
```

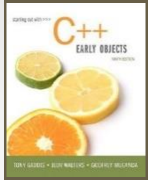
```
#include <iostream>
using namespace std;

void displayValue(const int &value);
void doubleValue(int &value);

int main()
{
   int x = 25;

   displayValue(x);
   return 0;
}

void displayValue(const int &value)
{
   doubleValue(value); // can't do this
   cout << "value is " << value << endl;
}

void doubleValue(int &value)
{
   value *= 2;
}
```

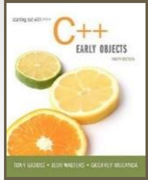


Constant Member Functions

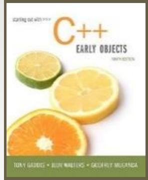
- We sometimes want to protect an object from being modified by a class function through the "this" pointer; we can do this by declared a const member function:

```
int Rectangle::doSomething() const;
```



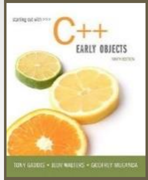


```
// declare the Rectangle class
class Rectangle
{
    private:
        int width = 0, height = 0;
    public:
        Rectangle() { width = 1; height = 1; }
        Rectangle(int width, int height) {
            this->width = width;
            this->height = height;
        }
        ~Rectangle() {}
        int getWidth() { return width; }
        int getHeight() { height = 20; return height; }
        void setWidth(int width) { this->width = width; }
        void setHeight(int height) { this->height = height; }
        int calculateArea();
};

int Rectangle::calculateArea() { return width * height; }
```



```
// declare the Rectangle class
class Rectangle
{
    private:
        int width = 0, height = 0;
    public:
        Rectangle() { width = 1; height = 1; }
        Rectangle(int width, int height) {
            this->width = width;
            this->height = height;
        }
        ~Rectangle() {}
        int getWidth() { return width; }
        int getHeight() const { height = 20; return height; }
        void setWidth(int width) { this->width = width; }
        void setHeight(int height) { this->height = height; }
        int calculateArea();
};

int Rectangle::calculateArea() { return width * height; }
```

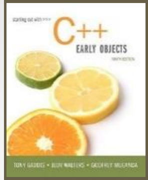
```
In member function 'int Rectangle::getHeight() const':
error: assignment of member 'Rectangle::height' in read-only object
Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s))
```



Static Member Variables

- Just as we declared static local variables, we can declare static member variables and functions which are shared by all objects of the class type

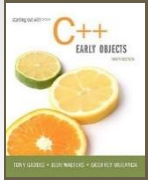
```
class StatDemo
{
        private:
            static int x;  // declaration only, need to allocate below
            int y;
            ...
};

int StatDemo::x; // allocation
```



```
#include <iostream>
using namespace std;

class StatDemo
{
    private:
        static int x;  // declaration only, need to allocate below
        int y;
    public:
        void setX(int x) { this->x = x; }
        void setY(int y) { this->y = y; }
        int getX() const { return this->x; }
        int getY() const { return this->y; }
};

int StatDemo::x; // allocation

int main()
{
    StatDemo obj1, obj2;
    obj1.setX(5);
    obj1.setY(10);
    obj2.setY(20);
    cout << "x: " << obj1.getX() << " " << obj2.getX() << endl;
    cout << "y: " << obj1.getY() << " " << obj2.getY() << endl;

    return 0;
}
```

```
x: 5 5
y: 10 20
```

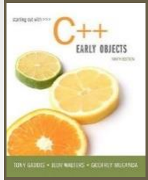
Static Member Functions

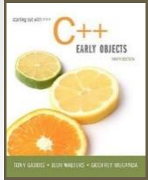


```
#include <iostream>
#include <sstream>
using namespace std;

class StatDemo
{
    private:
      static int x;  // declaration only, need to allocate below
      int y;
    public:
      void setX(int x) { this->x = x; }
      void setY(int y) { this->y = y; }
      int getX() const { return this->x; }
      int getY() const { return this->y; }
      static const string xToString();
};

int StatDemo::x; // allocation

// convert static variable x to a string
const string StatDemo::xToString()
{
    ostringstream convert;
    convert << x;
    return convert.str();
}
```

```
int main()
{
    StatDemo obj1, obj2;
    obj1.setX(5);
    obj1.setY(10);
    obj2.setY(20);
    cout << "x: " << obj1.getX() << " " << obj2.getX() << endl;
    cout << "y: " << obj1.getY() << " " << obj2.getY() << endl;
    cout << "static call xToString = " << StatDemo::xToString() << endl;
    cout << "non-static call xToString = " << obj2.xToString() << endl;
    return 0;
}
```

```
x: 5 5
y: 10 20
static call xToString = 5
non-static call xToString = 5
```

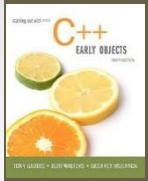




Static Members: Summary

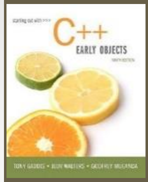
- Static members are variables or functions associated with a class
  - Can be accessed through the class name with "::" notation or using object instances
  - Use access specifiers (e.g. private) to prevent direct access to variables
- Instance members are non-static variables or functions associated with an object instance
  - Can only be accessed through an object instance

Friends of Classes

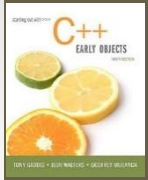


- A friend is a function that is not a member of a class but has access to private members of a class

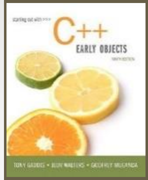
```
// Budget class declaration
class Budget
{
private:
    static double corpBudget;
    double divBudget;
public:
    Budget() { divBudget = 0; }
    void addBudget(double b)
       { divBudget += b; corpBudget += divBudget; }
    double getDivBudget() const { return divBudget; }
    static double getCorpBudget() { return corpBudget; }
    static void mainOffice(double);
    friend void Aux::addBudget(double b);
};
```



```
// Aux class declaration.
class Aux
{
private:
   double auxBudget;
public:
   Aux() { auxBudget = 0; }
   void addBudget(double b);
   double getDivBudget() const { return auxBudget; }
};
```

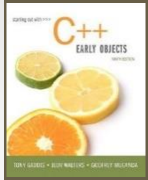


```
// auxil.cpp
void Aux::addBudget(double b)
{
   auxBudget += b;
   Budget::corpBudget += auxBudget;
}

// budget.cpp
double Budget::corpBudget = 0;

//**********************************************************
// Definition of static member function mainOffice        *
// This function adds the main office's budget request to  *
// the corpBudget variable.                                *
//**********************************************************
void Budget::mainOffice(double budReq)
{
   corpBudget += budReq;
}
```



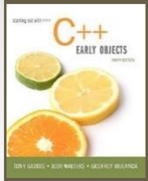
Inheritance

- Inheritance allows us to represent the "is-a" relationship in our classes, e.g.
  - a poodle is a dog
  - a car is a vehicle
  - a rectangle is a shape
- A base class is the general class (dog, vehicle, shape)
- A derived class is the specialized class (poodle, car, rectangle)
  - The derived class inherits attributes from the base class



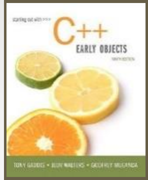
```
class Person {
    private:
        string name;
    public:
        Person() {
            this->name = "noname";
        }
        Person(string name) {
            this->name = name;
        }
        string getName() {
            return this->name;
        }
};
```

```
class Student : public Person {
    private:
        string major;
    public:
        Student() {
            this->major = "nomajor";
        }
        Student(string major) {
            this->major = major;
        }
        string getMajor() {
            return this->major;
        }
};
```



```
#include <iostream>
using namespace std;

...

int main()
{
   Person *p = new Person("Sam Jones");
   Student *s = new Student("Math");

   cout << p->getName() << endl;
   //cout << p->getMajor() << endl;  // can't do this!

   cout << s->getName() << endl;
   cout << s->getMajor() << endl;

   delete s;
   delete p;

   return 0;
}
```

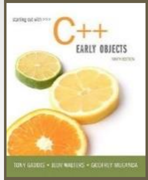
```
Sam Jones
noname
Math
```



Constructing the Base Class

- New Student class with modified constructor which calls parent constructor

```
class Student : public Person {
    private:
        string major;
    public:

        // constructors
        Student() {
            this->major = "nomajor";
        }

        Student(string name, string major) : Person(name) {
            this->major = major;
        }

        // accessors
        string getMajor() {
            return this->major;
        }
};
```



```
int main()
{
   Student *s = new Student("Leslie Knope", "Math");
   cout << s->getName() << ", " << s->getMajor() << endl;
   delete s;
   return 0;
}
```

```
Leslie Knope, Math
```