Creative Software Design

Class

Yunho Kim
yunhokim@hanyang.ac.kr
Dept. of Computer Science

Today's Topics

- Class and Instance
- Class access control
- Member functions
- Constructor, Destructor
- *this* pointer
- Struct in C vs. Struct in C++, Struct vs. Class in C++

Class

- A *class* is a user-defined data type,
  - which holds its own *member variables* and *member functions*.
  - These members can be accessed by creating an *instance* of that class.

```
class ClassName
{
accessSpecifier:
    memberVariables;
    ...
    MemberFunctions() {...}
    ...
...
};
```

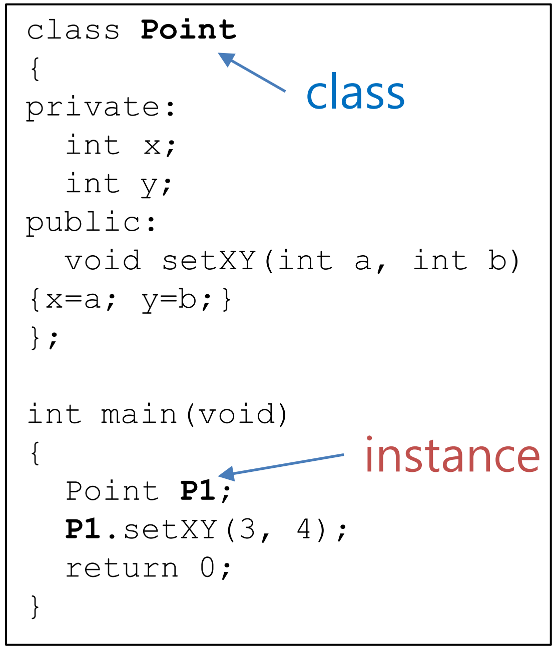
```
class Point
{
private:
  int x;
  int y;
public:
  void setXY(int a, int b) {x=a; y=b;}
};
```

- C++ classes are similar to C structs,
  - except member functions and other small differences.

```
typedef struct _Point
{
  int x;
  int y;
} Point;
```

Class vs. Instance / Object

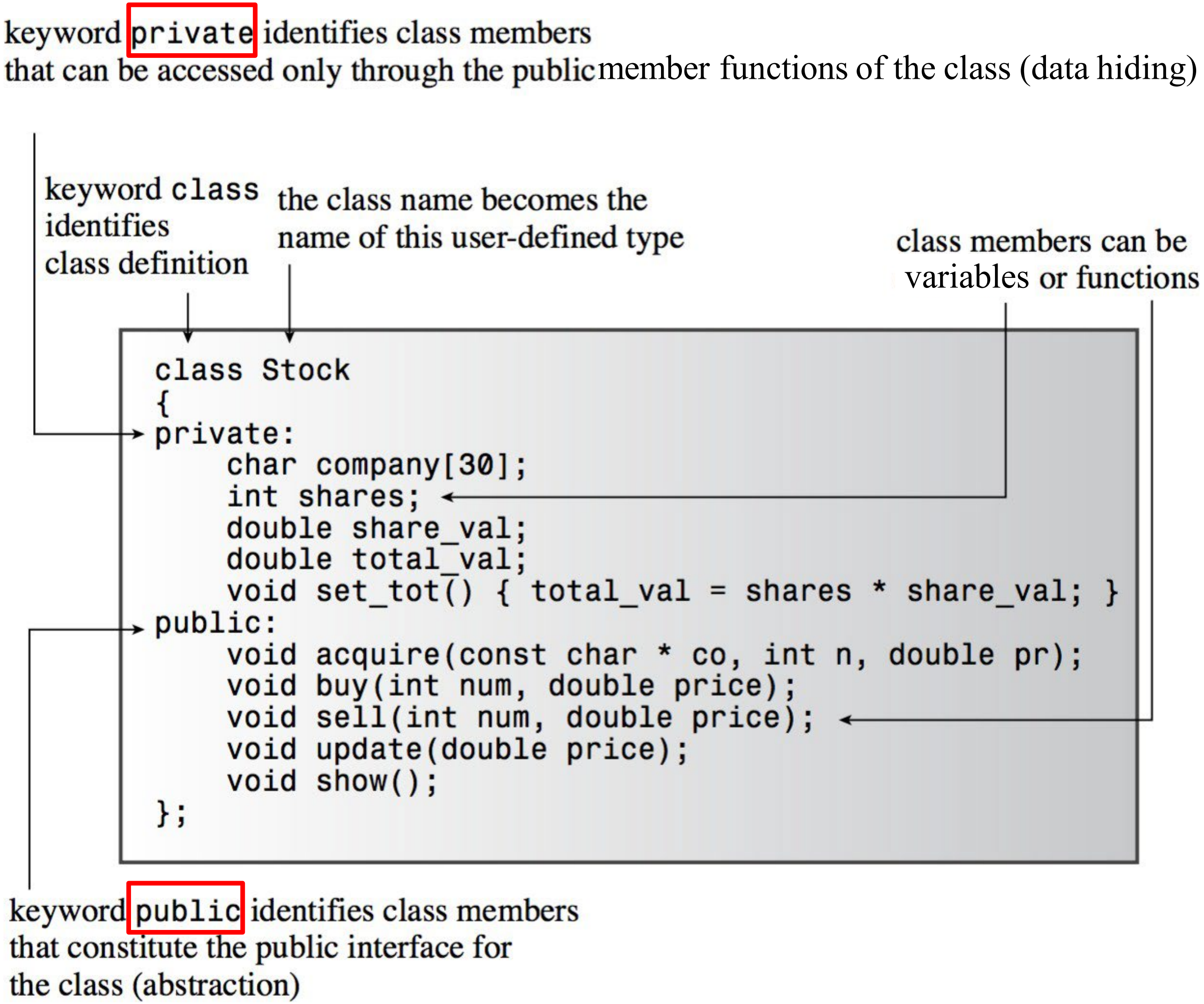
- Class - type vs. Instance (or Object) – variable
- Analogous to bread pan vs. bread.



```
class Point
{
private:
  int x;
  int y;
public:
  void setXY(int a, int b)
{x=a; y=b;}
};

int main(void)
{
  Point P1;
  P1.setXY(3, 4);
  return 0;
}
```

- Instantiation : creation of an **instance / object** of the **class**.
  - Instances have allocated memory to store specific data.
  - There can be multiple identical instances of the same class type, but there cannot exist identical classes.

Class Definition



keyword private identifies class members that can be accessed only through the public member functions of the class (data hiding)

keyword class identifies class definition

the class name becomes the name of this user-defined type

class members can be variables or functions

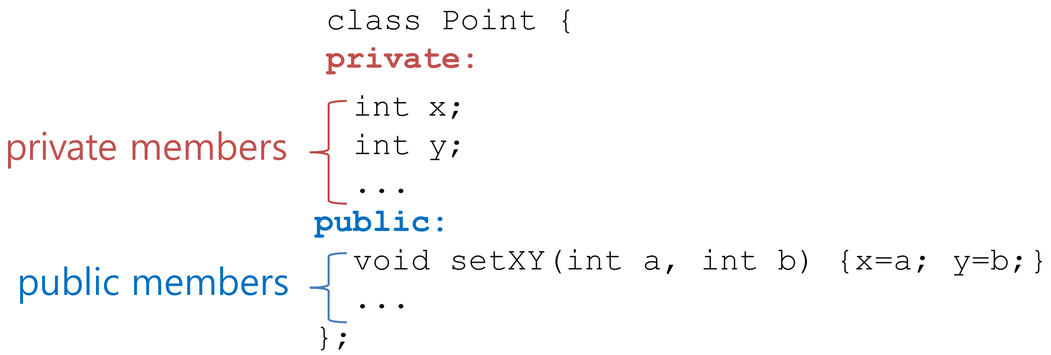
```
class Stock
{
private:
    char company[30];
    int shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }
public:
    void acquire(const char * co, int n, double pr);
    void buy(int num, double price);
    void sell(int num, double price);
    void update(double price);
    void show();
};
```

keyword public identifies class members that constitute the public interface for the class (abstraction)

Class Access Control

- Classes can have members with different access control.
  - The members are either `public`, `private`, or `protected` (*access specifiers*).
  - `public` members are accessible from anywhere.
  - `private` members are only accessible by its member functions.
  - `protected` members are accessible by its member functions and its derived classes' member functions - *will be covered in a later lecture (Inheritance).*

- Any member *encountered after a specifier* will have the associated access *until another specifier is encountered.*



```
class Point {
private:
    int x;
    int y;
    ...
public:
    void setXY(int a, int b) {x=a; y=b;}
    ...
};
```

private members

public members

Class Access Control

- If member variables are **`private`**, they **are not accessible outside of the class**.
  - They need **public** ***access functions***.

```
class Point {
private:
  int x;
  int y;
public:
  void setXY(int a, int b) {x=a; y=b;}
};
int main(void){
  Point P1;
  P1.x = 3; // compile error!
  P1.setXY(3, 4);
  return 0;
}
```

Class Access Control : Student Example

```
class Student {
 private:
  string name_, id_, grade_;
  int midterm_, final_, hw1_, hw2_;

 public:
  void SetInfo(string name, string id) { name_ = name, id_ = id; }
  void SetScores(int midterm, int final, int hw1, int hw2) {
    midterm_ = midterm, final_ = final, hw1_ = hw1, hw2_ = hw2;
  }
  void ProcessGrade() { ... }
  string GetGrade()   { return grade_; }
};

int main() {
  Student a_student;
  a_student.SetInfo("gdhong", "13001");
  a_student.SetScores(99, 90, 85, 100);
  a_student.ProcessGrade();  // Call the member function ProcessGrade.

  a_student.grade_ = "E-";  // Compile error!
  string grade = a_student.GetGrade();  // Fine.
  ...
}
```

Member Function

- A class can have member functions which work on the member variables of the class.
    - Member functions are **declared in the class definition**.
    - Member functions are **defined** either **in the class definition (in header files)** or **outside of the class definition (usually in source files).**
    - Member functions are accessed by using **. operator**, like member variables.

Member Function Definition in the Class Definition: Student Example

```
// student.h
class Student {
 private:
  string name_, id_, grade_;
  int midterm_, final_, hw1_, hw2_;

 public:
  void SetInfo(string name, string id)
  { name_ = name, id_ = id; }

  void SetScores(int midterm, int final, int hw1, int hw2)
  {
    midterm_ = midterm, final_ = final, hw1_ = hw1, hw2_ = hw2;
  }

  string GetGrade() { return grade_; }
};
```

Member Function Definition in the Class Definition: Student Example

```
// student.h
class Student {
private:
  string name_, id_, grade_;
  int midterm_, final_, hw1_, hw2_;

 public:
  void SetInfo(string name, string id);
  void SetScores(int midterm, int final, int hw1, int hw2);
  string GetGrade();
};
```

```
// student.cpp
#include "student.h"

void Student::SetInfo(string name, string id)
{ name_ = name, id_ = id; }

void Student::SetScores(int midterm, int final, int hw1, int hw2)
{
  midterm_ = midterm, final_ = final, hw1_ = hw1, hw2_ = hw2;
}

string Student::GetGrade()
{ return grade_; }
```

Member Function: Scope Resolution Operator (::)

- :: is used to specify the namespace or the class membership.
- A::B means B is in a namespace/class A.
- ::B means B belongs the global namespace (most C library).

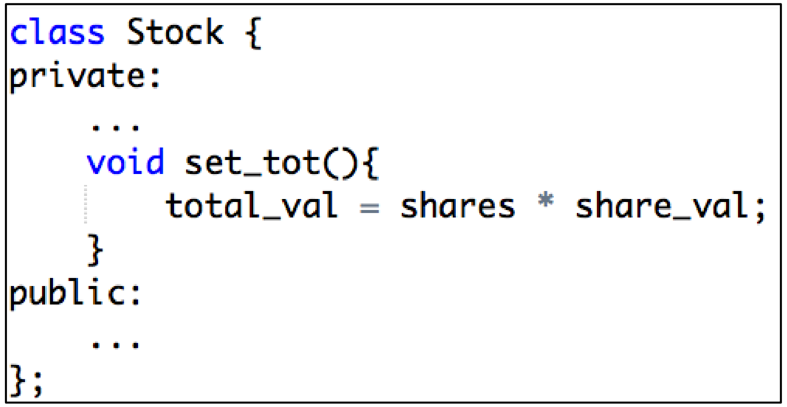
```
#include <cmath>
namespace my_namespace {

class MyClass {
  void FunctionA(int i);
  // ...
};

void MyClass::FunctionA(int i) { /* ... */ }
void FunctionB(double v, MyClass* a) { /* ... */ }

}  // namespace my_namespace

int main() {
  my_namespace::MyClass a;
  my_namespace::FunctionB(1.25, &a);
  double v = ::cos(0.0);
  return 0;
}
```

Quiz #1

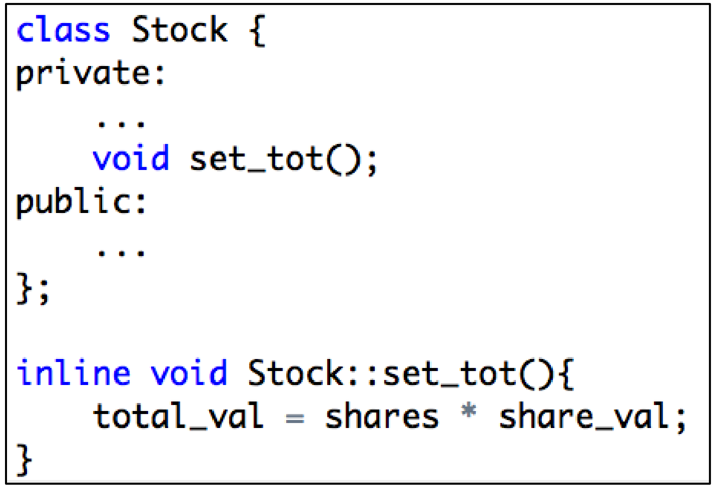
- Why do most of the member variables are declared as private?

Inline Member Functions

- To make a member function inline, you can define a member function in the class definition (in header file)
- Or you can define a member function outside the class definition (in header file) and use the *inline* qualifier
- Functions defined in source files cannot be inlined.
    - The definition of an inline function must be reachable in the translation unit where it is accessed



```
class Stock {
private:
    ...
    void set_tot(){
        total_val = shares * share_val;
    }
public:
    ...
};
```



```
class Stock {
private:
    ...
    void set_tot();
public:
    ...
};

inline void Stock::set_tot(){
    total_val = shares * share_val;
}
```

Inline Member Functions

- Question: Can I define a non-inline member function in a header file (outside the class definition)?
- Let's say main.cpp and test.cpp include one of the following header files:

```
#include <string>

class Student {
private:
    std::string name_;
public:
    std::string getName();
};

std::string Student::getName()
{
    return name_;
}
```

link error: multiple definition of Student::getName()

```
#include <string>

class Student {
private:
    std::string name_;
public:
    std::string getName();
};

inline std::string Student::getName()
{
    return name_;
}
```

Ok

→ Functions defined in a header file must be inline, otherwise you'll get multiple definitions error.

Constructor

- Constructors are special member functions that **initialize** the object and is called when the object **is created**.
- They have **the same name** as the class and **no return type**.
- They are automatically called when the object of its class type is instantiated.

```
class Student {
public:
  string name_, id_, grade_;
  ...
public:
  Student() { name_="noname"; id_="noid"; }
  ...
};

int main()
{
  Student st;  // Student::Student() is called!
  cout << st.name_ << endl;
}
```

Constructor Overloading

- A class can have multiple constructors.

```
class Student {
public:
   string name_, id_, grade_;
   ...
public:
   Student() { name_="noname"; id_="noid"; }
   Student(string name, string id) { name_=name; id_=id; }
   ...
};

int main()
{
   Student st1;    // Student::Student() is called!
   Student st2("Tom", "2016123456“); // Student::Student(string, string) is called!
}
```

Default Constructor

- A default constructor is a constructor which is called with no argument.
- Member variables that are not initialized in a constructor...
  - remain uninitialized (for primitive types such as `int`)
  - or initialized by calling their classes’ default constructor (for class types)

```
class Student {
public:
  string name_, id_, grade_;
  int midterm_, final_, hw1_, hw2_;
  ...
public:
  Student() // default constructor
  { name_="noname"; id_="noid"; }

  Student(string name, string id)   // this is not a default constructor
  { name_=name; id_=id; }

// member variables other than name_ & id_ remain...
// uninitialized (for primitive types, e.g., midterm_)
// or initialized by their classes’ default constructor (for class type,
e.g., grade_ will be initialized by calling std::string::string() )
  ...
};
```

Default Constructor

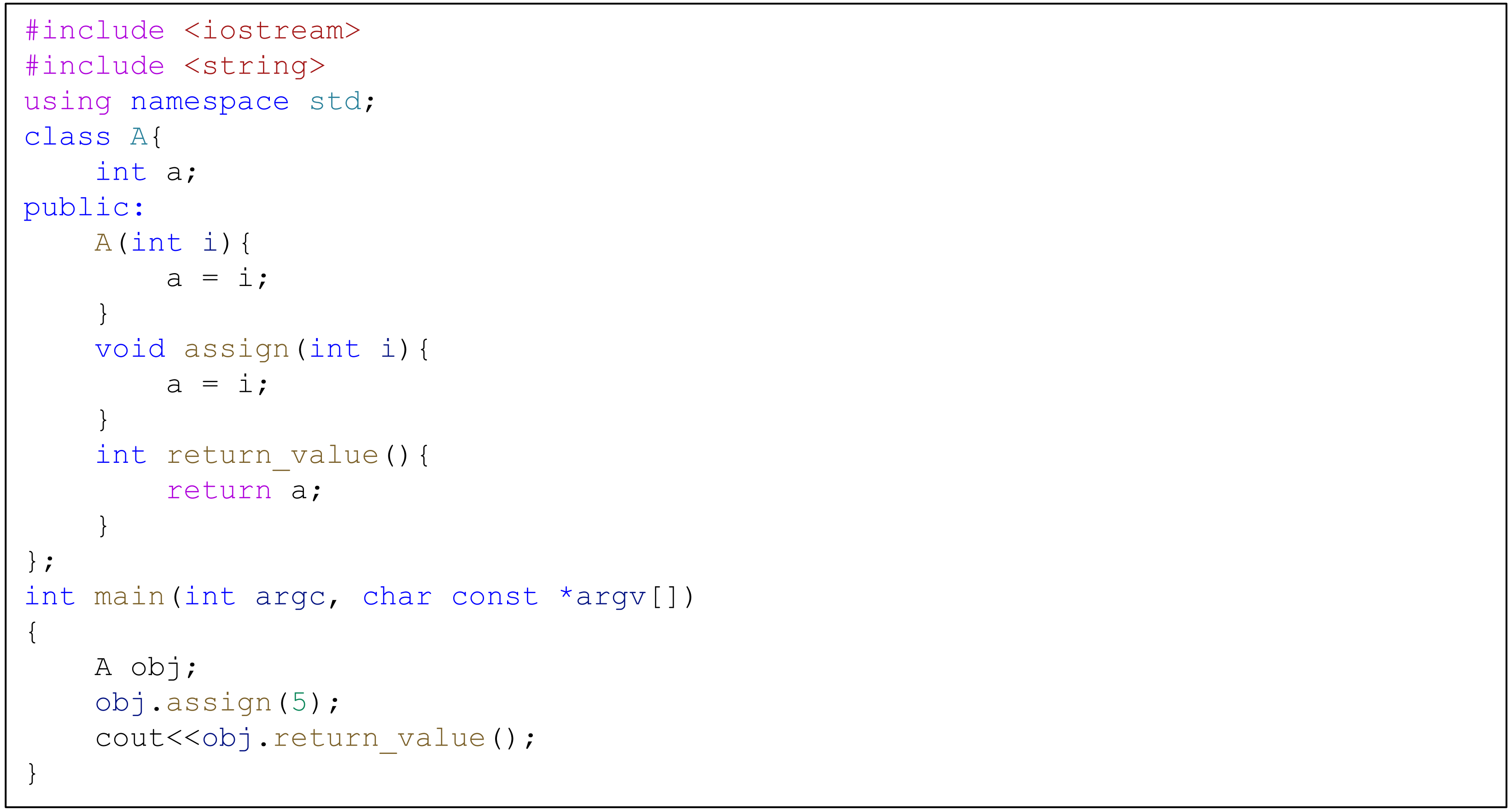
- A **default constructor** is implicitly created by compiler if there is no user-defined constructor.

```
class Stock
{
public:
    string company;
    long shares;
    double share_val;
};

int main()
{
    Stock stock; // implicitly declared
default constructor is called!

    cout << stock.company << endl;
    cout << stock.shares << endl;  c
    out << stock.share_val << endl;
    return 0;
}
```

```
class Stock
{
public:
    string company;
    long shares;
    double share_val;

    Stock(const string& co, long n, double pr)
    {}
};

int main()
{
    Stock stock;  // compile error!

    cout << stock.company << endl;
    cout << stock.shares << endl;
    cout << stock.share_val << endl;
    return 0;
}
```

Quiz #2

- What is the expected output? (including compile/runtime error)



```
#include <iostream>
#include <string>
using namespace std;
class A{
    int a;
public:
    A(int i){
        a = i;
    }
    void assign(int i){
        a = i;
    }
    int return_value(){
        return a;
    }
};
int main(int argc, char const *argv[])
{
    A obj;
    obj.assign(5);
    cout<<obj.return_value();
}
```

Constructor Member Initializer List

- Member initializer list is the place where non-default initialization of member variables can be specified.
  - Members of primitive type (such as int) are initialized with the parameter.
  - Members of class type is initialized **by calling the proper constructor** taking the parameter.

```
class Stock
{
public:
    string company;
    const long shares;
    double share_val;

    Stock(const string& co, long n, double pr)
    : company(co), shares(n), share_val(pr)
    { // shares = n causes a compile error
    }
};
```

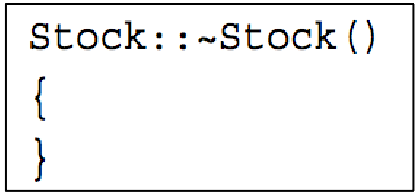
Operator new and Class Constructor

- T* p = new T;
  - If T is a primitive type: Allocates memory space to store data of type T
  - If T is a class: Allocates memory space and initialize it
  - **by calling default constructor of T**

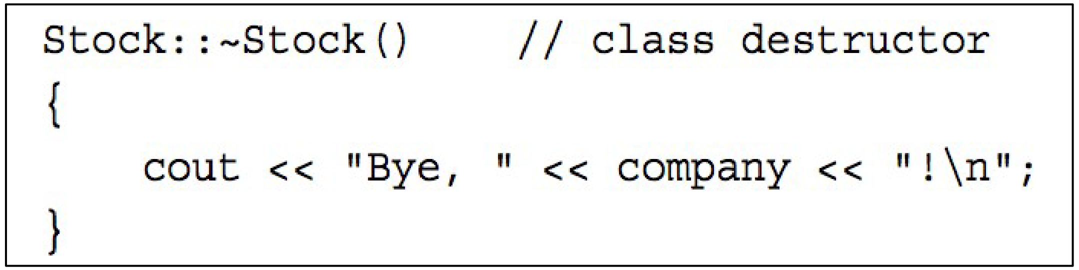
- T* p = new T(*arguments*);
  - If T is a primitive type: Allocates memory space and initialize it with the *arguments*
  - If T is a class: Allocates memory space and initialize it
  - **by calling the proper constructor that takes *argument***

Destructor

- A destructor is a special member function **for clean-up** that is **called when the object is destructed.**
- Its name is '~' + the class name.
- It has no arguments and no return type.



```
Stock::~Stock()
{
}
```



```
Stock::~Stock()     // class destructor
{
    cout << "Bye, " << company << "!\n";
}
```

Destructor Example

(Focus on ~DoubleArray() destructor!)

```
class DoubleArray {
 public:
  DoubleArray() : ptr_(NULL), size_(0) {}
  DoubleArray(size_t size) : ptr_(NULL), size_(0) { Resize(size); }

  ~DoubleArray() { if (ptr_) delete[] ptr_; }

  void Resize(size_t size);

  int size() const { return size_; }
  double* ptr() { return ptr_; }
  const double* ptr() const { return ptr_; }

 private:
   double* ptr_;
   size_t size_; // size_t is unsigned int.
};

void DoubleArray::Resize(size_t size) {
  double* new_ptr = new double[size];
  if (ptr_) {
    for (int i = 0; i < size_ && i < size; ++i) new_ptr[i] = ptr_[i];
    delete[] ptr_;
  }
  ptr_ = new_ptr;
  size_ = size;
}
```

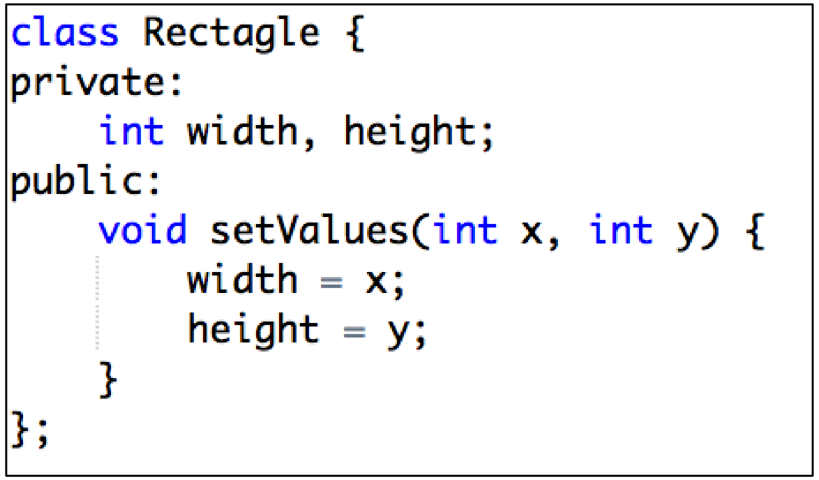
Quiz #3

- What is the expected output? (including compile/runtime error)

```
#include <iostream>
#include <string>
using namespace std;
class A{
public:
    A(){ cout<<"A's Constructor called\n";}
    ~A(){cout<<"A's Destructor called\n"; }
};
class B{
    A a;
public:
    B(){cout<<"B's Constructor called\n";}
    ~B(){cout<<"B's Destructor called\n";}
};
int main(int argc, char const *argv[]){
    B b1;
}
```

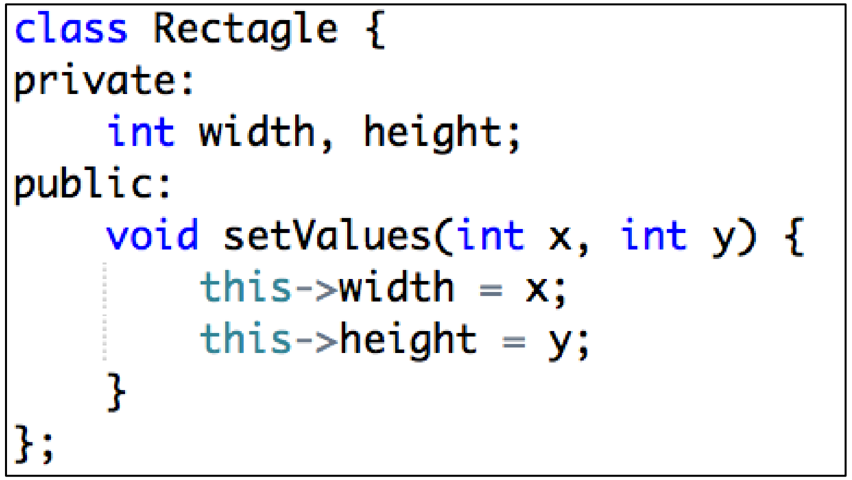
this Pointer

- Every object in C++ has access to its own address through a pointer called *this* pointer.
- *this* pointer points to the object used to invoke a member function or access to a member variable (passed as a hidden argument to the function).



```
class Rectagle {
private:
    int width, height;
public:
    void setValues(int x, int y) {
        width = x;
        height = y;
    }
};
```

=



```
class Rectagle {
private:
    int width, height;
public:
    void setValues(int x, int y) {
        this->width = x;
        this->height = y;
    }
};
```

Member Varriable and Parameter Names

- Question: Can member variables and function parameters have the same name?

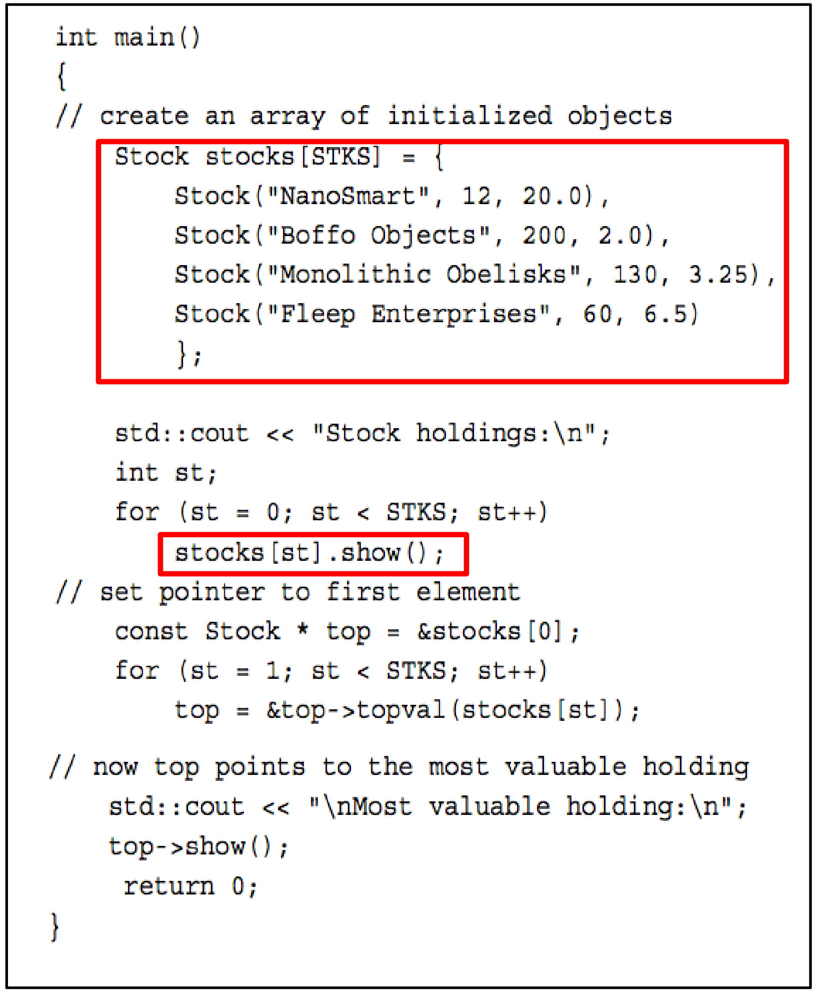
-> Yes, if you use the "this" pointer.

```
class Rect
{
public:
    int width, height;  Rect():
    width(1), height(2) {}
    void setValues(int width, int y)
    {
        this->width = width;
        height = y;
    }
};
int main()
{
    Rect rt;  rt.setValue
    s(10, 20);
    cout << rt.width << endl; // 10
    return 0;
}
```

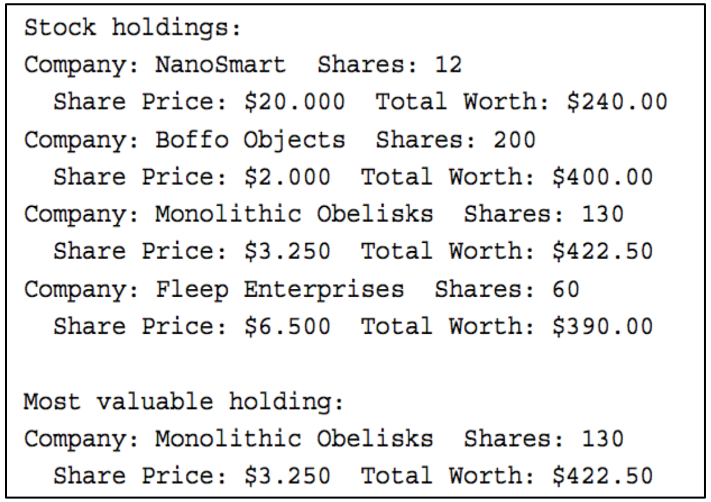
```
class Rect
{
public:
    int width, height;  Rect():
    width(1), height(2) {}
    void setValues(int width, int y)
    {
        width = width;
        height = y;
    }
};
int main()
{
    Rect rt;  rt.setValue
    s(10, 20);
    cout << rt.width << endl; // 1 ?
    return 0;
}
```

This is valid, but the result is not what you expect.
It's easy to make mistakes, so don't use it.

Array of Objects



```
int main()
{
// create an array of initialized objects
    Stock stocks[STKS] = {
        Stock("NanoSmart", 12, 20.0),
        Stock("Boffo Objects", 200, 2.0),
        Stock("Monolithic Obelisks", 130, 3.25),
        Stock("Fleep Enterprises", 60, 6.5)
        };

    std::cout << "Stock holdings:\n";
    int st;
    for (st = 0; st < STKS; st++)
        stocks[st].show();
// set pointer to first element
    const Stock * top = &stocks[0];
    for (st = 1; st < STKS; st++)
        top = &top->topval(stocks[st]);

// now top points to the most valuable holding
    std::cout << "\nMost valuable holding:\n";
    top->show();
     return 0;
}
```



```
Stock holdings:
Company: NanoSmart  Shares: 12
  Share Price: $20.000  Total Worth: $240.00
Company: Boffo Objects  Shares: 200
  Share Price: $2.000  Total Worth: $400.00
Company: Monolithic Obelisks  Shares: 130
  Share Price: $3.250  Total Worth: $422.50
Company: Fleep Enterprises  Shares: 60
  Share Price: $6.500  Total Worth: $390.00

Most valuable holding:
Company: Monolithic Obelisks  Shares: 130
  Share Price: $3.250  Total Worth: $422.50
```

Struct in C vs. Struct in C++

- In C, `struct` has only member variables, and is usually used with `typedef`
  - to avoid using `struct` keyword when declaring a variable (`struct _Point p1;`).
- In C++, `struct` has member variables and **member functions**, and **does not need `typedef`**.

```c
typedef struct _Point {
  int x;
  int y;
}Point;

int main(void){

  Point P1;
  P1.x = 3;
  P1.y = 4;
  return 0;
}
```

C

```cpp
struct Point {
  int x;
  int y;
  void setXY(int a, int b) {x=a; y=b;}
};

int main(void){

  Point P1;
  P1.x = 3;
  P1.y = 4;
  P1.setXY(1, 2);
  return 0;
}
```

C++

Struct in C vs. Struct in C++

- In C, all `struct` member variables are *public* (can be accessed from anywhere).
- In C++, `struct` members can be one of *public, private,* or *protected* (the default is *public*).

```
typedef struct _Point {
  int x;
  int y;
}Point;

int main(void){

  Point P1;
  P1.x = 3;
  P1.y = 4;
  return 0;
}
```

C

=

```
struct Point {
  int x;
  int y;
};

int main(void){

  Point P1;
  P1.x = 3;
  P1.y = 4;
  return 0;
}
```

C++

=

```
struct Point {
public:
  int x;
  int y;
};

int main(void){

  Point P1;
  P1.x = 3;
  P1.y = 4;
  return 0;
}
```

C++

Struct vs. Class in C++

- In C++, `struct` and `class` are almost the same.
- The only difference is default accessibility of members:
  - In `struct`, *public* is default
  - In `class`, *private* is default

```cpp
struct Point {
private:
  int x;
  int y;
public:
  void setXY(int a, int b) {x=a; y=b;}
};

int main(void){

  Point P1;
  P1.setXY(3, 4);
  return 0;
}
```

=

```cpp
class Point {
  int x;
  int y;
public:
  void setXY(int a, int b) {x=a; y=b;}
};

int main(void){

  Point P1;
  P1.setXY(3, 4);
  return 0;
}
```