Creative Software Design

6 – Class

Yoonsang Lee
Fall 2023

Midterm Exam

- Date & time: **Oct 31, AM 09:30 ~ 10:30**
- Place: **IT.BT 609**
- Scope: Lecture 2 ~ 7

- **You cannot leave until 30 minutes after the start of the exam** even if you finish the exam earlier.

- That means, **you cannot enter the room after 30 minutes from the start of the exam** (do not be late, never too late!).

- Please bring your **student ID card** to the exam.

- We will not accept questions unless the error in the problem is clearly evident. You should solve the problem based on the information provided in the question.

Outline

- Class & 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 structures,
  - except member functions and access control.

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

Class vs. Instance (or 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;
}
```

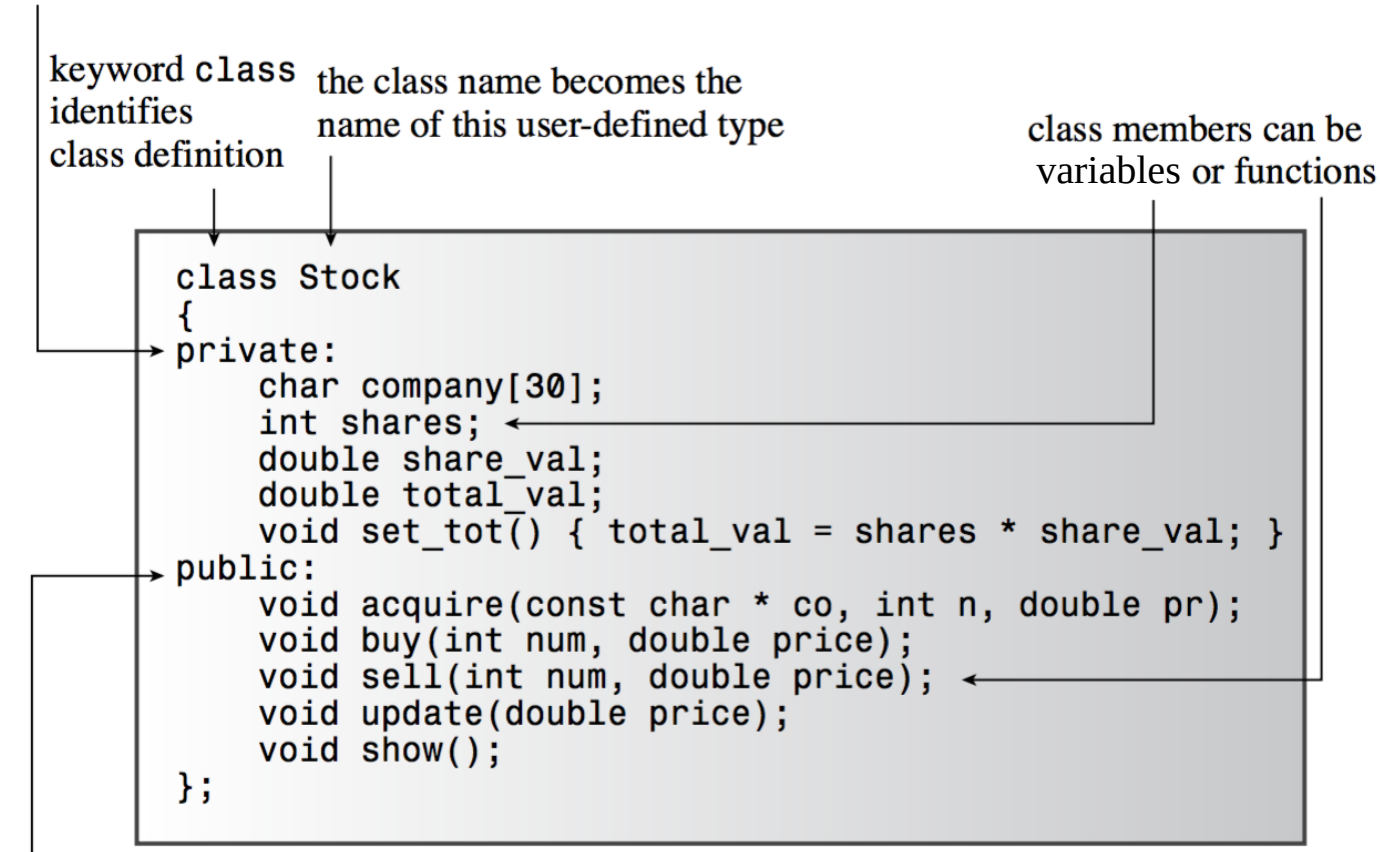
class

instance

- Creating an instance (or object) of a class is called *instantiation*.
- 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 member functions of the class (data hiding)



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 the class's member functions**.
  - **`protected`** members are **accessible by the class's member functions** and **its derived classes' member functions** - *will be covered in a later lecture (8-Inheritance).*

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

private members

public members

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

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 : Stock example

```
class Stock  // class declaration
{
private:
    std::string company;
    long shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }
public:
    void acquire(const std::string & co, long n, double pr);
    void buy(long num, double price);
    void sell(long num, double price);
    void update(double price);
    void show();
};    // note semicolon at the end
```

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_ = "D-";  // 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 varia bles.

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) // inline function
  { name_ = name, id_ = id; }

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

  string GetGrade() { return grade_; } // inline function
};
```

Member function definition outside of 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 funtions).

```
#include <math.h>
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;
}
```

Member function: Stock example

stock.cpp

```
void Stock::acquire(const std::string & co, long n, double pr)
{
    company = co;
    if (n < 0)
    {
        std::cout << "Number of shares can't be negative; "
                  << company << " shares set to 0.\n";
        shares = 0;
    }
    else
        shares = n;
    share_val = pr;
    set_tot();
}
```

stock.h

```
class Stock  // class declaration
{
private:
    std::string company;
    long shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }
public:
    void acquire(const std::string & co, long n, double pr);
    void buy(long num, double price);
    void sell(long num, double price);
    void update(double price);
    void show();
};    // note semicolon at the end
```

Member function: Stock example

stock.cpp

```
void Stock::sell(long num, double price)
{
    using std::cout;
    if (num < 0)
    {
        cout << "Number of shares sold can't be negative. "
             << "Transaction is aborted.\n";
    }
    else if (num > shares)
    {
        cout << "You can't sell more than you have! "
             << "Transaction is aborted.\n";
    }
    else
    {
        shares -= num;
        share_val = price;
        set_tot();
    }
}
```

stock.h

```
class Stock  // class declaration
{
private:
    std::string company;
    long shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }
public:
    void acquire(const std::string & co, long n, double pr);
    void buy(long num, double price);
    void sell(long num, double price);
    void update(double price);
    void show();
};    // note semicolon at the end
```

Quiz 1

- Go to https://www.slido.com/
- Join #**csd-ys**
- Click "Polls"

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that your quiz answer must be submitted **in the above format** to receive a quiz score!

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 (BUT IN header f ile) and use the *inline* qualifier.
- DO NOT define inline functions in source files.
  - If an inline function defined in a source file is called from another source file, you'll get 'unresolved e xternal' error (https://isocpp.org/wiki/faq/inline-functions#inline-member-fns).

```
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_;
}
```

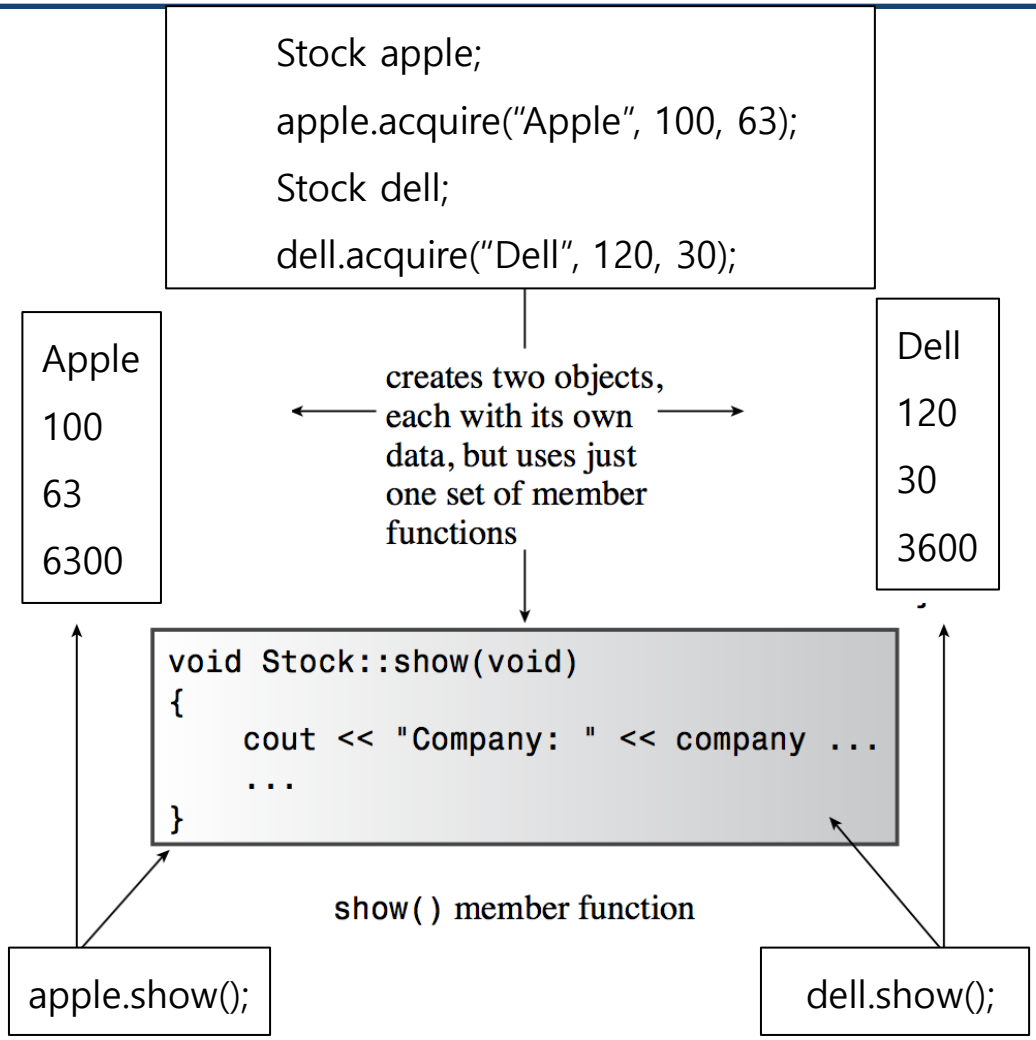
link error: multiple definition of Student::getName()

```
#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.

Class vs. Instance : Stock example 1



Class vs. Instance : Stock example 2

```
int main() {

    Stock apple;
    apple.acquire("Apple", 20, 12.50);
    apple.show();
    apple.buy(15, 18.125);
    apple.show();
    apple.sell(400, 20.00);
    apple.show();
    apple.buy(300000, 40.125);
    apple.show();
    apple.sell(300000, 0.125);
    apple.show();
    return 0;
}
```

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 defined.

```
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!
}
```

Constructor Overloading

- (Note) You can initialize a primitive type variable in a similar way.

```
int main()
{
  int a1 = 10;
  int a2(10);
}
```

Default constructor

- A default constructor is **a constructor which is called with no argument.**

```
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; }
};
```

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;
    cout << 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;
}
```

Constructor : Stock example

stock.cpp

```
Stock::Stock(const string & co, long n, double pr)
{
company = co;

    if (n < 0)
    {
        std::cerr << "Number of shares can't be negative; "
                    << company << " shares set to 0.\n";
        shares = 0;
    }
    else
        shares = n;
    share_val = pr;
    set_tot();
}
```

Quiz 2

- Go to https://www.slido.com/
- Join #**csd-ys**
- Click "Polls"

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that your quiz answer must be submitted **in the above format** to receive a quiz score!

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 arguments.**

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

    Stock(const string& co, long n, double pr)
    : company(co), shares(n), share_val(pr)
    {}
};
```

Member initializer list &default constructor

- For member variables that are not initialized in a constructor initializer list,
  - Members of primitive type (such as int) remain uninitialized.
  - Members of class type initialized by calling their classes' **default constructor.**

```
class Student {
public:
  string name_, id_, grade_;
  int midterm_, final_, hw1_, hw2_;
  ...
public:
  Student(string name, string id)
  : 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() )
  ...
};
```

Operator new and class constructor

- T* p = new T;
  - If T is a primitive type: Allocates memory space to store data of type T and the space remain uninitialized.
  - 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 *arguments***

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

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

    Stock() { cout << "Stock::Stock()" << endl; }

    Stock(const string& co, long n, double pr)
    : company(co), shares(n), share_val(pr)
    { cout << "Stock::Stock(const string&, long, double)" << endl; }
};

int main()
{
    int* i1 = new int;
    int* i2 = new int(10);

    Stock* s1 = new Stock;
    Stock* s2 = new Stock("Apple", 10, 125.0);

    delete i1;
    delete i2;
    delete s1;
    delete s2;

    return 0;
}
```

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.

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

- It has no arguments and no return type.

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

- When a destructor is called,
  - First, the body of the destructor function is executed.
  - Second, destructors for member objects are called.
    - (except static members → refer to Lecture 11)
  - (Third, destructors of base classes are called → refer to Lecture 8)

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;
}
```

Default Destructor

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

- For many classes, a default destructor is sufficient. You only need to define a custom destructor
  - when the class stores handles to system resources that need to be released,
  - or pointers that point to dynamically allocated memory locations, also need to be released.

Stock class example

Listing 10.4 **stock10.h**

```
// stock10.h -- Stock class declaration with constructors, destructor added
#ifndef STOCK10_H_
#define STOCK01_H_
#include <string>

class Stock
{
private:
    std::string company;
    long shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }
public:
// two constructors
    Stock();        // default constructor
    Stock(const std::string & co, long n = 0, double pr = 0.0);
    ~Stock();       // noisy destructor
    void buy(long num, double price);
    void sell(long num, double price);
    void update(double price);
    void show();
};

#endif
```

Stock class example

Listing 10.5 **stock10.cpp**

```
// stock10.cpp -- Stock class with constructors, destructor added
#include <iostream>
#include "stock10.h"

// constructors (verbose versions)
Stock::Stock()        // default constructor
{
    std::cout << "Default constructor called\n";
    company = "no name";
    shares = 0;
    share_val = 0.0;
    total_val = 0.0;
}

Stock::Stock(const std::string & co, long n, double pr)
{
    std::cout << "Constructor using " << co << " called\n";
    company = co;

    if (n < 0)
    {
        std::cout << "Number of shares can't be negative; "
                  << company << " shares set to 0.\n";
        shares = 0;
    }
    else
        shares = n;
    share_val = pr;
    set_tot();
}
// class destructor
Stock::~Stock()        // verbose class destructor
{
    std::cout << "Bye, " << company << "!\n";
}
```

Stock class example

Listing 10.6 **usestok1.cpp**

```
// usestok1.cpp -- using the Stock class
// compile with stock10.cpp
#include <iostream>
#include "stock10.h"

int main()
{
  {
    using std::cout;
    cout << "Using constructors to create new objects\n";
    Stock stock1("NanoSmart", 12, 20.0);            // syntax 1
    stock1.show();
    Stock stock2 = Stock ("Boffo Objects", 2, 2.0); // syntax 2
    stock2.show();

    cout << "Assigning stock1 to stock2:\n";
    stock2 = stock1;
    cout << "Listing stock1 and stock2:\n";
    stock1.show();
    stock2.show();

    cout << "Using a constructor to reset an object\n";
    stock1 = Stock("Nifty Foods", 10, 50.0);    // temp object
    cout << "Revised stock1:\n";
    stock1.show();
    cout << "Done\n";
  }
    return 0;
}
```

Quiz 3

- Go to https://www.slido.com/
- Join #**csd-ys**
- Click "Polls"

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that your quiz answer must be submitted **in the above format** to receive a quiz score!

this pointer

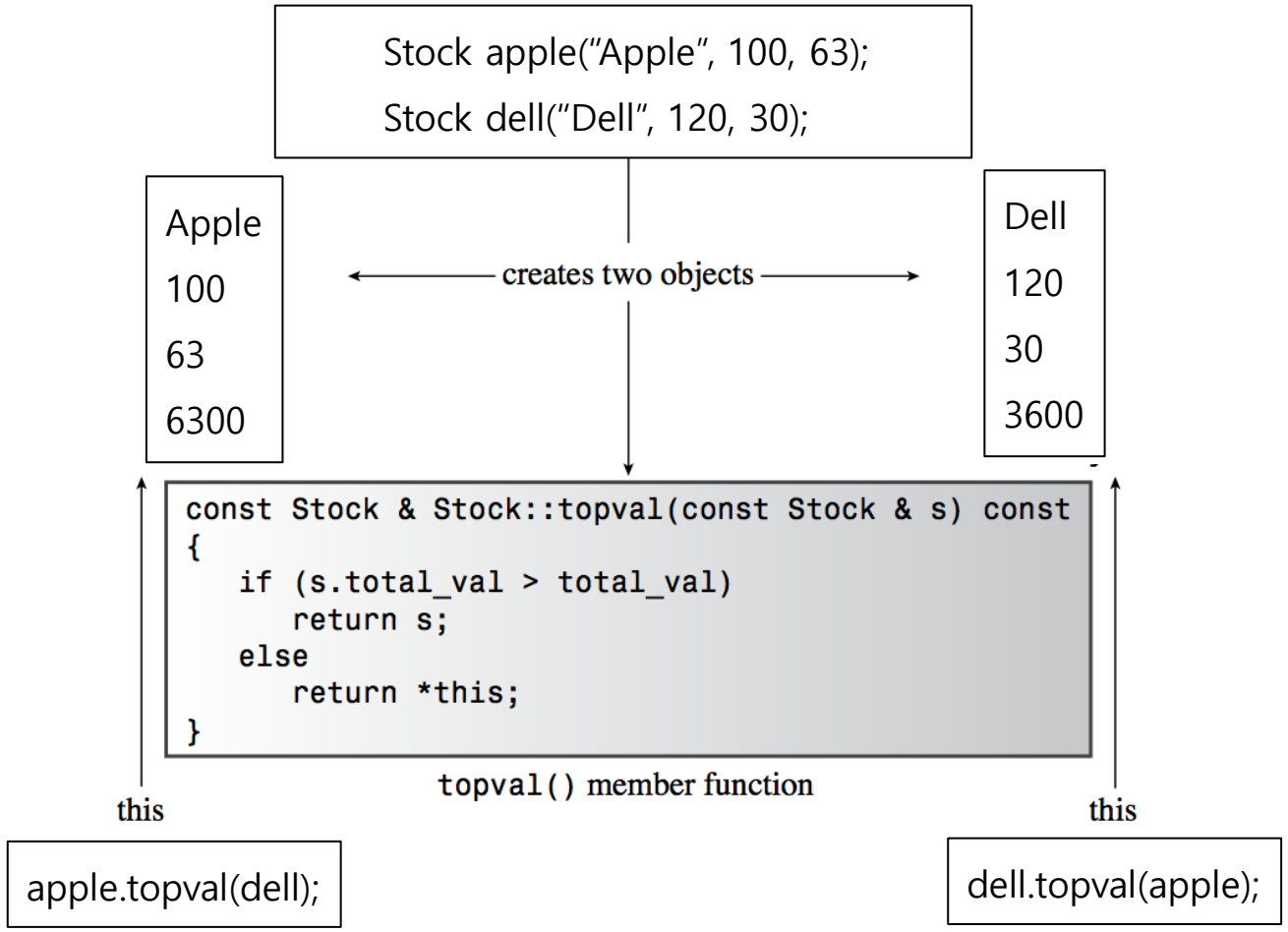
- Every object in C++ has access to its own address through a pointer called *this* pointer.
- *"this* pointer" points to "this object", used to invoke a member function or access to a member variable (passed as a hidden argument to the member f unction)

```
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;
    }
};
```

this pointer – returning self reference



Invokes topval() with apple,
so *this* points to apple:
s is dell, **this* is apple

Invokes topval() with dell,
so *this* points to dell:
s is apple, **this* is dell

Member Var. and Parameter Names

- Question: Can member variables and function parameters have the same name? -> Yes, but it's not recommended.

```
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.setValues(10, 20);
    cout << rt.width << endl; // 10
    return 0;
}
```

if you always use the “this” pointer, that would be okay.

```
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.setValues(10, 20);
    cout << rt.width << endl; // 1
    return 0;
}
```

But if you don't, it compiles and runs fine, but the result will not be what you think.

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 `type def`
  - to avoid using `struct` keyword when declaring a variable (`struct _Point p1;`).
- In C++, `struct` has member variables and **member functions**, and **do es 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

```c
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 an ywhere).
- In C++, `struct` members can be one of *public*, *private*, or *protected* (t he default is *public*).

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

int main(void){

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

C

=

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

int main(void){

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

C++

=

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

int main(void){

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

C++

Struct in C vs. Struct in C++

- If members are *private* in C++ `struct`, they are not accessible outside of the class. They need public *access functions*.

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

int main(void){

  Point P1;
  P1.x = 3; // compile error!
  P1.y = 4; // compile error!
  return 0;
}
```

```
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;
}
```

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

```
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;
}
```

=

```
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;
}
```

Next Time

- Labs for this lecture:
  - Lab1: Assignment 6-1
  - Lab2: Assignment 6-2

- Next lecture:
  - 7 - Standard Template Library