Creative Software Programming

6 – Class

Today's topics

- Struct in C / C++, Struct vs. Class in C++
- Class access control
- Member variable and function
- Class vs. Instance
- Constructor, destructor
- *this* pointer

Structured information

- Informations of students taking a class

```
Name     ID       Grade    Midterm  Final    HW1      HW2
-------- -------- -------- -------- -------- -------- --------
gdhong   13001    A+       99       90       85       100
cskim    13002    A        80       95       93       90
yhlee    13003    B+       85       80       92       88
...
```

- How can you represent these data ?

Structured information

- One option is to use array

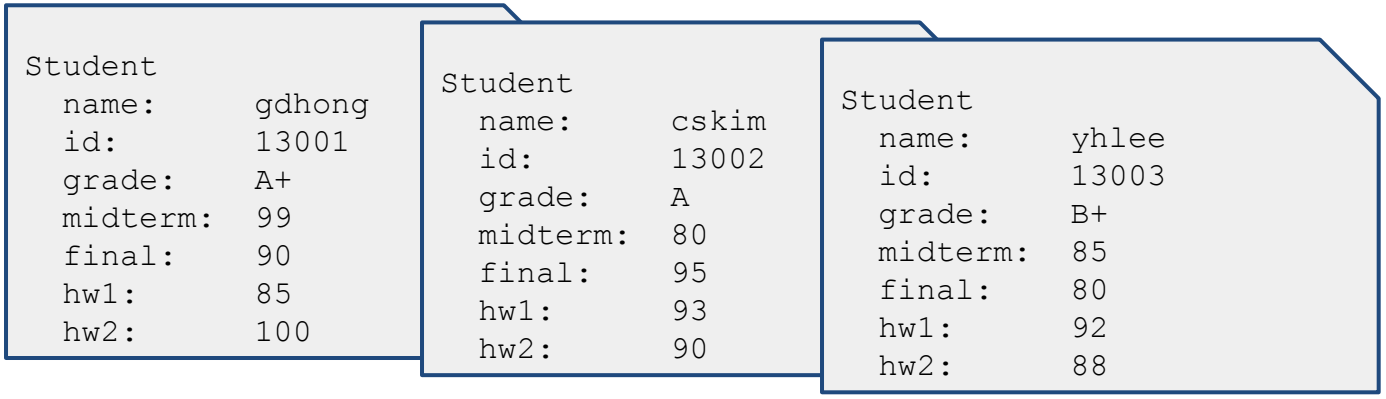
Name	ID	Grade	Midterm	Final	HW1	HW2
gdhong	13001	A+	99	90	85	100
cskim	13002	A	80	95	93	90
yhlee	13003	B+	85	80	92	88
...						

```
void ProcessGrade(int num_students, const string* names, const string* ids,
                  const int* midterm, const int* final,
                  const int* hw1, const int* hw2, string* grades) {
  for (int i = 0; i < num_students, ++i) {
    int sum = midterm[i] + final[i] + hw1[i] + hw2[i];
    if (sum >= 95) grades[i] = "A+";
    else if (sum >= 90) grades[i] = "A";
    else if (sum >= 85) grades[i] = "B+";
    ...
  }
}
```

Structured information

- Another option is to use struct

Name	ID	Grade	Midterm	Final	HW1	HW2
gdhong	13001	A+	99	90	85	100
cskim	13002	A	80	95	93	90
yhlee	13003	B+	85	80	92	88
...						



Structured information

- Another option is to use struct

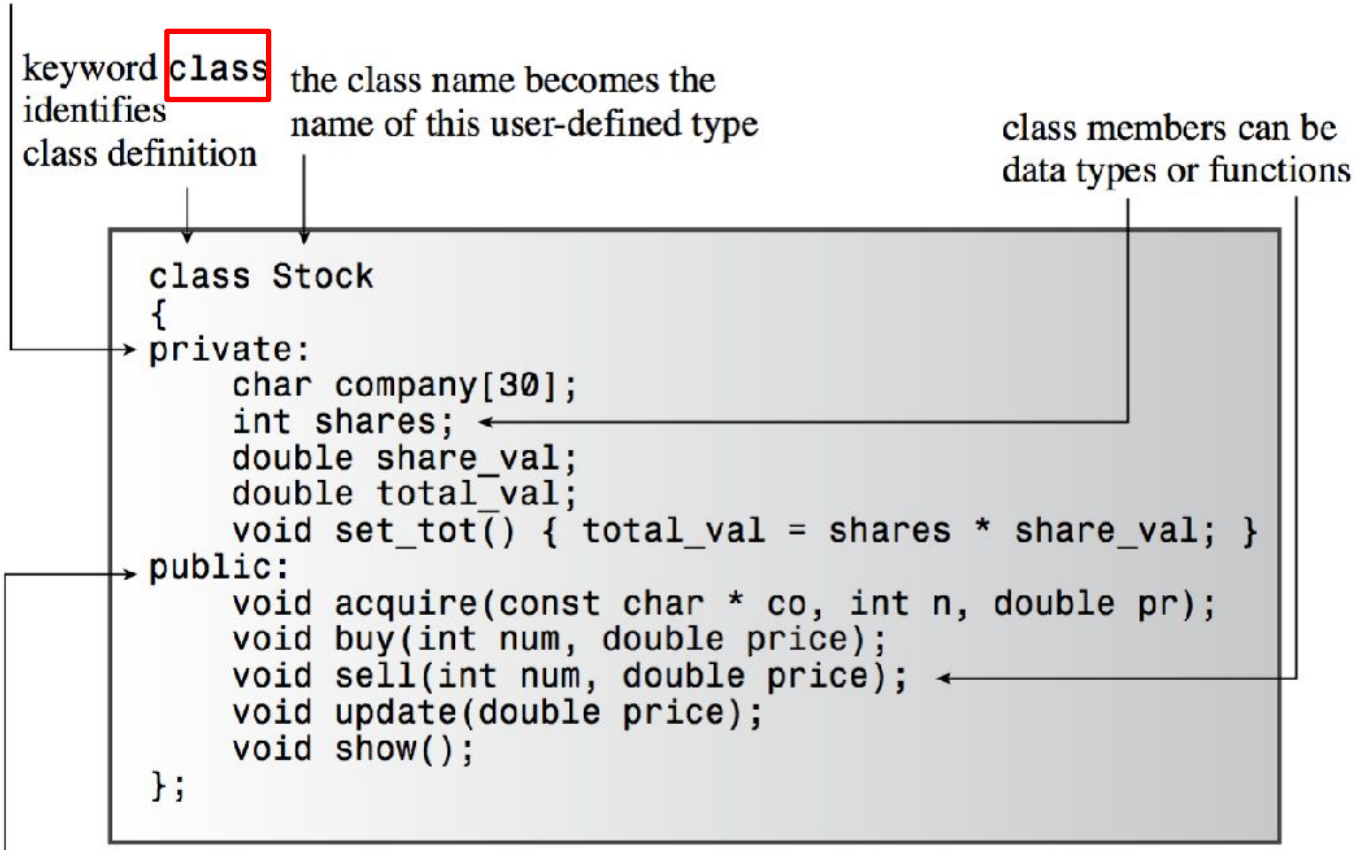
Name	ID	Grade	Midterm	Final	HW1	HW2
gdhong	13001	A+	99	90	85	100
cskim	13002	A	80	95	93	90
yhlee	13003	B+	85	80	92	88
...						

```
struct Student {
  string name, id, grade;
  int midterm, final, hw1, hw2;
};

void ProcessGrade(Student* students, int num_students) {
  for (int i = 0; i < num_students, ++i) {
    int sum = students[i].midterm + students[i].final +
        students[i].hw1 + students[i].hw2;
    if (sum >= 95) students[i].grade = "A+";
    else if (sum >= 90) students[i].grade = "A";
    else if (sum >= 85) students[i].grade = "B+";
    ...
  }
  ...
}
```

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 member variables with different access control.
  - The members are either `public`, `private`, or `protected`.
  - `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.
  - For a `class`, default is `private`,
    whereas for a `struct`, default is `public`.

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.
  ...
}
```

Struct in C / 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 / C++

- In C, all struct member variables are *public* (can be accessed from anywhere).
- In C++, struct member variables / functions can be *public, private, protected* (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
class Point {
 public:
  int x;
  int y;
};

int main(void){
  Point P1;
  P1.x = 3;
  P1.y = 4;
  return 0;
}
```

C++

Member function

- Classes can have member functions.
  - 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 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_; }
```

:: - 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 <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

```
#include <iostream>
#include <string>

using namespace std;

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

int main() {
  Stock s;
  s.company = "Apple";
  cout << s.company << endl;
  return 0;
}
```

- What is the expected output of the this program?
  - 1) Apple
  - 2) company
  - 3) A compile error occurs

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 source file) and use the inline qualifier

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

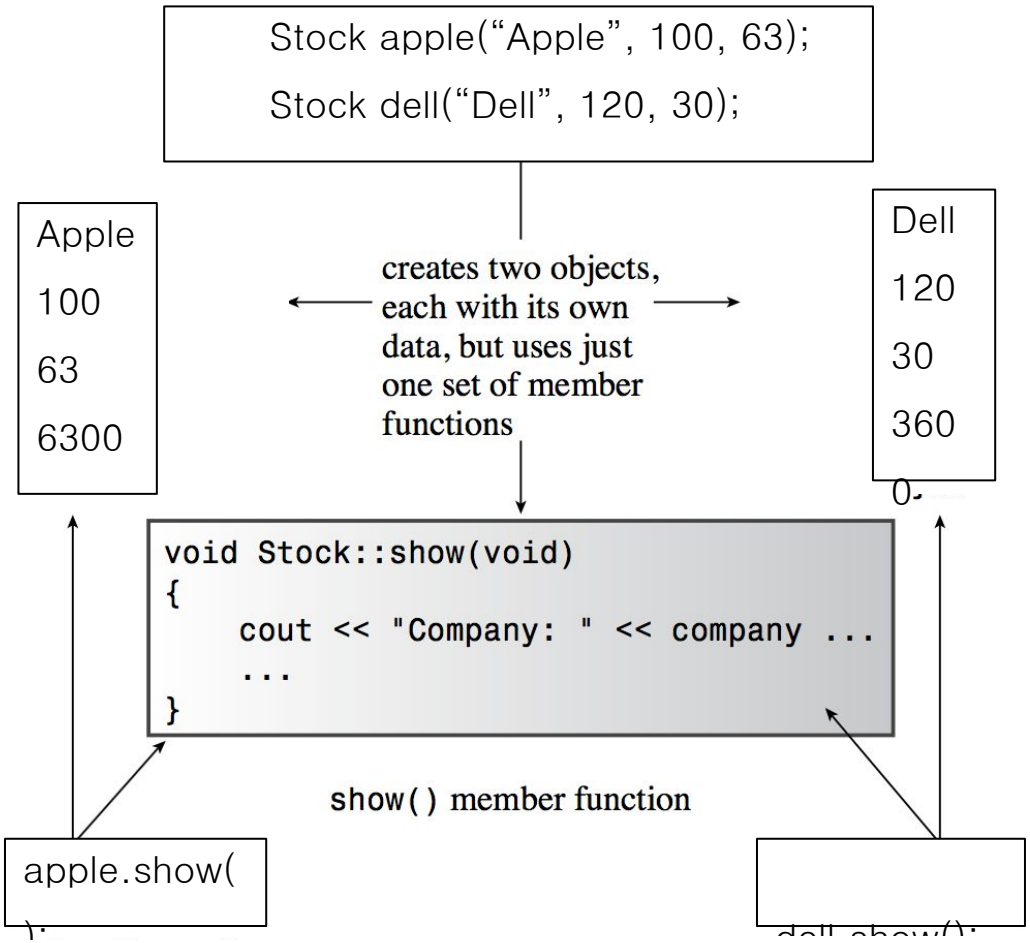
```
class Stock {
private:
    ...
    void set_tot();
public:
    ...
};

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

Class vs. Instance

- class - type, instance - variable
- Analogous to bread pan vs. bread.
- Instantiation : making 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 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.
- They have the same name as the class and no return type.
- They are automatically called when the object of its class type is declared.

```
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_ 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;
  cout << stock.share_val << endl;
  return 0;
}
```

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

  Stock(const string& c, long n, double p) {}
};

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

- What is the expected output of the this program?
  - 1) Apple 20
  - 2) Dell 10
  - 3) A compile error occurs

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

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

 public:
  Stock() {
    company = "Dell";
    shares = 10;
    share_val = 10.1;
  }

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

int main() {
  Stock s;
  cout << s.company << " " << s.shares << endl;
  return 0;
}
```

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;
  long shares;
  double share_val;

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

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*

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

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

  Stock() {}

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

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.
- 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;
}
```

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
```

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";
}
```

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

- What is the expected output of the following program? (If a compile error is expected, just write down "error").

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

class A {
 public:
  A() { cout << "a1 "; }
  ~A() { cout << "a2 "; }
};

class B {
 public:
  B() { cout << "b1 "; }
  ~B() { cout << "b2 "; }
};

void test() {
  B b;
}

int main() {
  A a;
  test();
  return 0;
}
```

this pointer

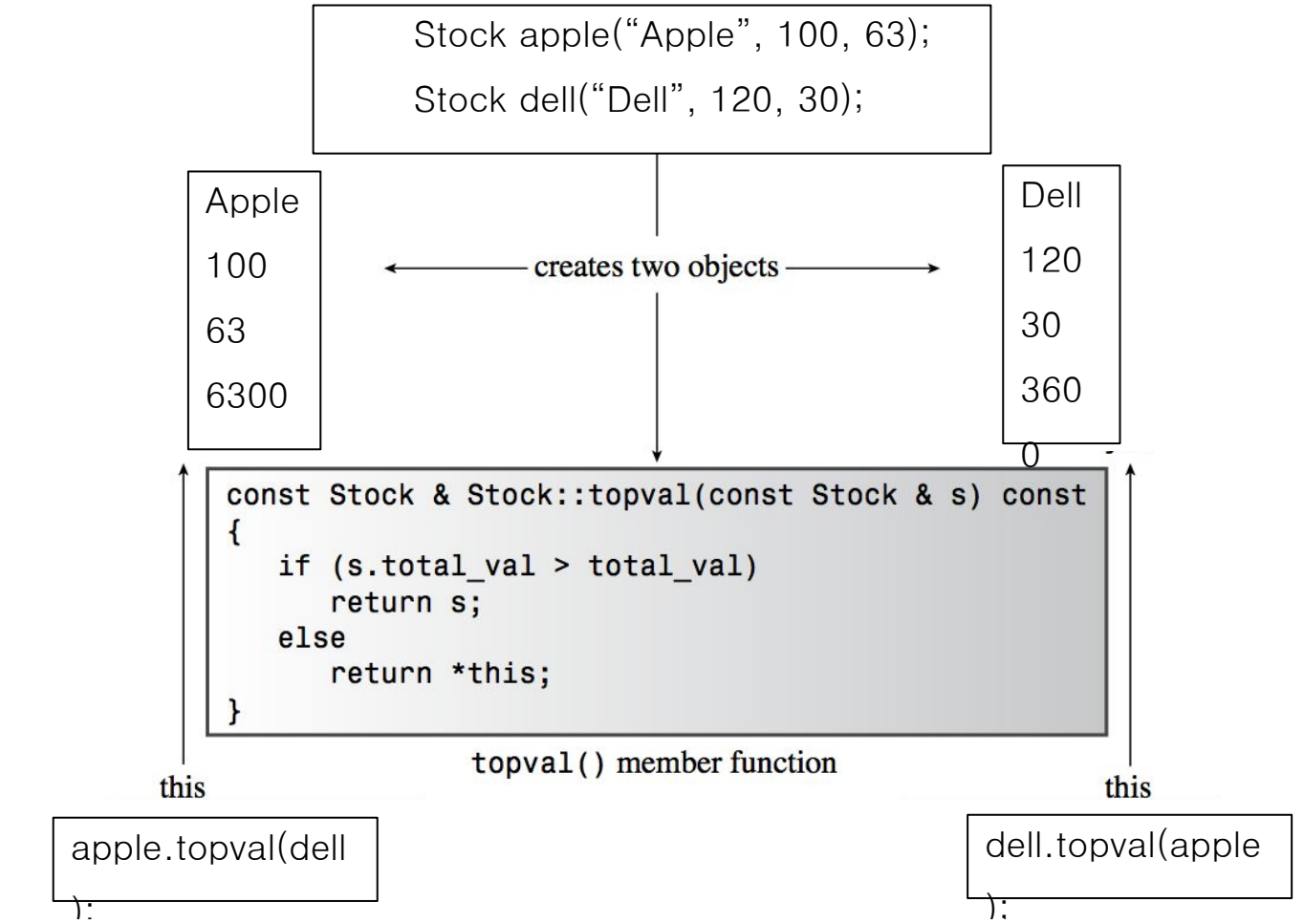
- Every object in C++ has access to its own address through a pointer called *this* pointer.
- *this* pointer points to the object used to invoke a member function or access to a member variable (passed as a hidden argument to the function).

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

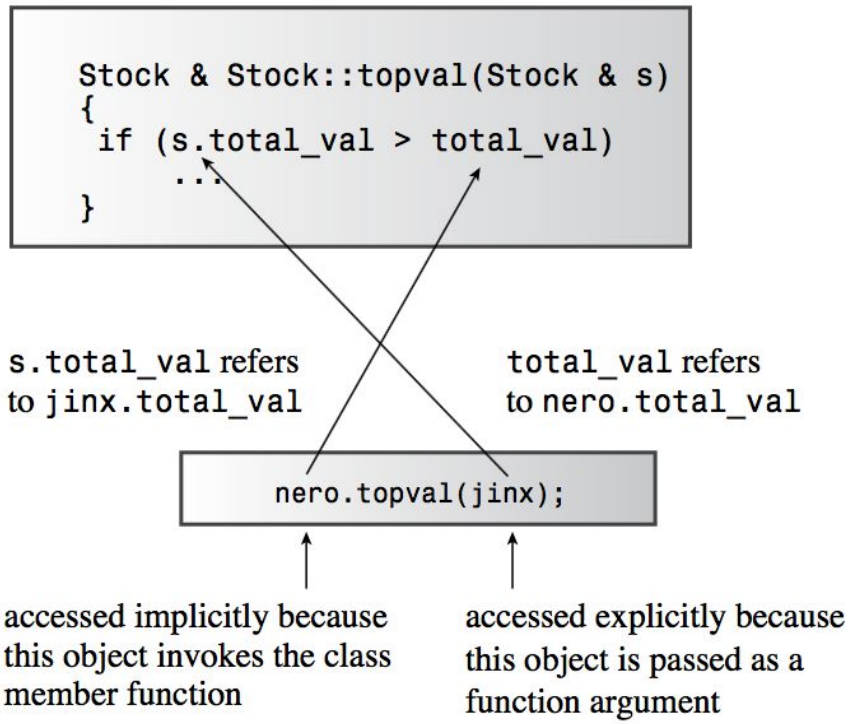
=

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

this pointer – returning self reference



this pointer



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
```

Next Time

- Next lecture:
  - 7 - Standard Template Library