

More Classes



C++ Object Oriented Programming
Pei-yih Ting
NTOU CS

..... 1

Contents

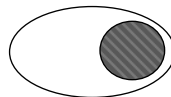
- ❖ Object composition and constructors
- ❖ Initialization of object within object
- ❖ Returning pointers
- ❖ **this** pointer
- ❖ Exploiting implicit references
- ❖ Class conversion
- ❖ Static data members
- ❖ Static member functions

2

Object Component

❖ Sometimes you would like to use a well designed object as a component to help accomplishing the task

❖ In that case, we have an object within another object



❖ Example:

```

class Person {
public:
    Person(const char *name);
    ~Person();
    char *getName() const;
private:
    char *m_name;
};

class DormRoom {
public:
    DormRoom(const char *myName,
              const char *roommateName);
    void listPeople() const;
private:
    Person m_me;
    Person m_roommate;
};

void main() {
    DormRoom *myRoom;
    myRoom = new DormRoom("Jamie", "Paul");
    myRoom->listPeople();
    delete myRoom;
}

DormRoom::DormRoom(const char *myName,
                    const char *roommateName) {
}

NOT working!!
error C2512: 'Person' : no appropriate default
                    constructor available

```

3

Solving The Initialization Problem

❖ First try: not working, call Person ctor within DormRoom ctor, i.e.

```

DormRoom::DormRoom(const char *myName, const char *roommateName) {
    m_me(myName);
    m_roommate(roommateName);
}

```

❖ Second try: not a good one, require default ctor, extra CPU time, depending on some uncertain factors

```

DormRoom::DormRoom(const char *myName, const char *roommateName) {
    m_me = Person(myName);
    m_roommate = Person(roommateName);
}

```

❖ Third try: a safe and syntactically legal solution, but undesirable

```

class Person {
...
    Person();
    void setName(const char *name);
};

```

❖ Correct solution: using initialization list

```

DormRoom::DormRoom(const char *myName, const char *roommateName)
    : m_me(myName), m_roommate(roommateName) {
}

```

4

Returning Pointers

- ❖ The function getName() violates *data encapsulation*

```
class Person {
public:
    Person(const char *name);
    ~Person();
    char *getName() const;
private:
    char *m_name;
};
```

- ❖ Why? Consider the following code: looks OK

```
void DormRoom::listPeople() const {
    cout << "I," << m_me.getName() << ", live in this room along with my roommate "
        << m_roommate.getName() << "\n";
}
```

- ❖ What would happen if it were written like this

```
void DormRoom::listPeople() const {
    char *tempString = m_me.getName();
    tempString[0] = '#';
    cout << "I," << tempString << ", live in this room along with my roommate "
        << m_roommate.getName() << "\n";
}
```

Interfering the integrity of the private data of Person class

5

Solution to Data Encapsulation Problem

- ❖ Simple solution provided by the grammar to prevent **incidental** breaking of the encapsulation

```
class Person {
public:
    Person(const char *name);
    ~Person();
    const char *getName() const;
private:
    char *m_name;
};
```

unintentional

Won't be able to mutate the content of m_name within this member function

```
const char *Person::getName() const {
    return m_name;
}
```

```
void DormRoom::listPeople() const {
    const char *tempString = m_me.getName();
    // tempString[0] = '#'; // compiler rejects this statement
    cout << "I," << tempString << ", live in this room along with my roommate "
        << m_roommate.getName() << "\n";
}
```

- ❖ Other solutions? use a string object

6

this pointer

- ❖ In the first C++ translator, by Stroustrup, C++ functions is translated to pure C functions. How can a function access some variables (those member variables) not defined in that function? Ex.

```
class Grades {
public:
    Grades(int score);
    int getScore();
private:
    int m_score;
};

int Grades::getScore() {
    return m_score;
}

void main() {
    Grades student1(95), student2(85), student3(45);
    cout << student1.getScore();
    cout << student2.getScore();
    cout << student3.getScore();
}
```

which variable is this referring to

- ❖ The compiler generates an *implicit* reference to the object which called the function and passes it into the function as an argument.

- ❖ Explicitly referencing the object

```
int Grades::getScore() {
    return this->m_score;
}
```

7

The primary purpose of *this* pointer

- ❖ The *this* pointer is most commonly used when objects need to be linked to other objects

```
class LinkedList {
public:
    void insert(LinkedList *newNode);
private:
    LinkedList *previous;
    LinkedList *next;
};
```

- ❖ We want to insert a new node into the list after another object

with `currentObject->insert(newObject);`

- ❖ The actual way to achieve the goal is using this pointer

```
void LinkedList::insert(LinkedList *newNode) {
    newNode->next = next; // implicitly referring the member of current object
    newNode->previous = this; // or next->previous
    next->previous = newNode;
    next = newNode;
}
```

8

Exploiting Implicit References

✧ Suppose we want to add a function to class **Grades** that checks if two objects contain the same score

✧ Here is the call in main()

```
if (grade1.equal(grade2))
    cout << "same scores";
else
    cout << "different scores";
```

✧ Here is the function

```
bool Grades::equal(Grades &secondScore) {
    return m_score == secondScore.m_score;
}
```

✧ Do not ignore implicit dereferencing

```
bool Grades::equal(Grades &firstScore, Grades &secondScore) {
    return firstScore.m_score == secondScore.m_score;
}
```

Note how clumsy the call is to this function

```
if (grade1.equal(grade1, grade2))
    ....
```

Type Conversion Constructor

✧ Suppose we would like to convert raw minutes to Time object

```
class Time {
public:
    Time();
    Time(int hours, int minutes, int seconds);
    Time(int rawMinutes);
private:
    int m_hours;
    int m_minutes;
    int m_seconds;
    void normalize();
};

Time::Time(): m_seconds(0), m_minutes(0), m_hours(0) {}

Time::Time(int hours, int minutes, int seconds)
    : m_hours(hours), m_minutes(minutes), m_seconds(seconds) {
    normalize();
}

Time::Time(int rawMinutes): m_seconds(0), m_minutes(rawMinutes), m_hours(0) {
    normalize();
}

void Time::normalize() {
    m_minutes += m_seconds / 60;
    m_seconds = m_seconds % 60;
    m_hours += m_minutes / 60;
    m_minutes = m_minutes % 60;
    m_hours = m_hours % 24;
}
```

Type Conversion Constructor

✧ Usage:

```
void main() {
    int x = 125;
    Time object;
    object = Time(125); // temporary object, assignment operator
    object = 125;
    object = x;
    object = (Time) x;
}
```

implicit invocation of type conversion ctor, construct a temporary object, assignment operator

Class Conversion

```
class Celsius; // forward declaration
class Fahrenheit {
public:
    Fahrenheit(int temperature);
    Fahrenheit(Celsius &cTemperature);
    int getTemperature() const;
    void display() const;
private:
    int m_temperature;
};

Fahrenheit::Fahrenheit(Celsius &cTemperature) {
    int celsiusTemperature = cTemperature.getTemperature();
    m_temperature = (int)(9.0 * celsiusTemperature / 5 + 32.5);
}

class Celsius {
public:
    Celsius(int temperature);
    Celsius(Fahrenheit &fTemperature);
    int getTemperature() const;
    void display() const;
private:
    int m_temperature;
};
```

Usage:

```
Fahrenheit room(75);
Celsius zimmer(18);
room = zimmer;
```

Static Data Members

✧ Suppose we want to give each object of the Student class a unique ID

✧ Using a global variable is one method

```
int gIDNumber = 0;
class Student {
public:
    Student();
    int getID() const;
private:
    int m_id;
};
```

✧ The constructor

```
Student::Student():m_id(gIDNumber++) {
}
```

✧ Problems:

- * If other programs manipulate this global variable, the count would be incorrect
- * It would be better if a name like gStudentIDNumber is used

13

Static Data Members (cont'd)

✧ Better solution with static data member

```
class Student {
public:
    Student();
    int getID() const;
private:
    static int lastIDNumber;
    int m_id;
};
```

✧ A class declaration is not a variable, you must define the static variable in the global scope

```
int Student::lastIDNumber = 0;
```

this can be put anywhere in the program, but it must be in the *.cpp file and only occurs once

✧ The constructor

```
Student::Student():m_id(lastIDNumber++) {
}
```

✧ Also used for specific constant definition. Ex. Integer::INT_MAX

14

Static Member Functions

✧ A static function can only access static data member

```
class Student {
public:
    Student();
    int getID() const;
private:
    static int lastIDNumber;
    int m_id;
    static int getNewID();
    static int incrementNewID();
};
```

✧ The keyword static is not repeated in the function definition

```
int Student::getNewID() {          int Student::incrementNewID() {
    return lastIDNumber;          return lastIDNumber++;
}                                  }
```

✧ The constructor might take this form

```
Student::Student():m_id(getNewID()) {
    incrementNewID()
}
```

15

Static Member Functions (cont'd)

✧ If the static member function is public, it can be accessed without reference to a particular object, ex.

```
Integer::convertFromInt(10);
```

✧ Static member function does not have the implicit *this* pointer because it is not invoked with any object.

✧ Sometimes use static member functions to implement callback functions that do not allow any implicit argument.

16