**Chapter 13**

**Additional Features of Classes**

**Overview**

Consider the following class declaration (see Example10):

class Cards

{

private:

int noCards;

int topCard;

int\* deck;

public:

Cards();

Cards(int noCards);

~Cards();

int getNoCards();

int dealACard();

void shuffle();

void displayCard(int cardNo, int x, int y);

};

**Parameters with same name as private data field**

Consider the following Constructor

Cards::Cards(int noCards)

{

this->noCards = noCards;

topCard = 0;

deck = new int[noCards];

}

What if we implement the Constructor this way:

Cards::Cards(int noCards)

{

noCards = noCards;

topCard = 0;

deck = new int[noCards];

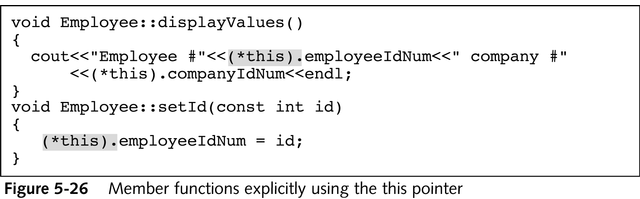
}

Does not assign anything to the pdf noCards - it is hidden!

If the formal parameter has the same name as the pdf, can use scope operator or implicit this pointer as shown.

The implicit this pointer is used only **within** the method. It refers to the address of the object that the method is invoked on. The this pointer is used for resolving ambiguity - the datafield preceded by this-> is always the private data field.

Can also dereference the this pointer in example below:



What are the advantages classes provide over structured programming (e.g. struct and functions)?

Information Hiding

Encapsulation (Private data/Public Methods)

Inheritance

Run-time Polymorphism

**Dynamically allocated pdfs**

Consider the deck pdf. Note how deck is dynamically allocated in the Constructor:

deck = new int[noCards];

**Destructors**

Destructor - cleanup dynamically allocated pointers that are pdfs. The Destructor has the following properties:

1. Named after class and preceded by ~ (Example: ~Cards)

2. Only one per class allowed

3. No input parameters allowed.

4. Automatically invoked when object goes out of score (when return statement

encountered).

Destructor is necessary for cleaning up those datafields that are dynamically allocated pointers. Example implementation for Cards class is shown below:

Cards::~Cards()

{

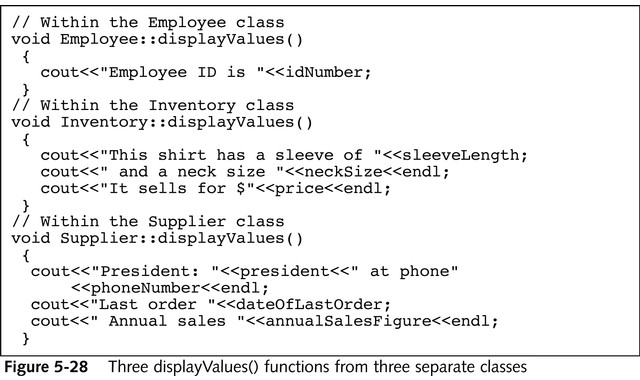
delete[] deck;

}

The Destructor is invoked when the object goes out of scope. See Example from class.

**Polymorphism**

* C++ allows you to create three very different methods, one for each of the classes in question
* When you can apply the same method name to different objects, your programs become easier to read and make more sense



**Overloading Methods**

* Multiple methods within a class can have the same method name too.
* Consider the constructors for the Rectangle class:
  + Rectangle();
  + Rectangle(int w, int h);
* Overloaded methods consist of same name, different parameters/number of parameters.
* Method signature: number of parameters, data type of the parameters, and the name of the method

Overloaded methods (for an angle class):

void setAngle(double angle\_in\_deg);

void setAngle(float angle\_in\_deg);

void setAngle(double radians, int factor);