**Chapter 14**

**Inline Classes/Copy Constructor/Containment**

**Inline Methods**

* It is possible to implement the methods in your .h (header) file (when declaring the class) instead of in a separate .cpp file.
* Often used for short methods (like setters/getters/Constructors/Destructors)
* Provides a way for declaring/implementing methods in one place

Example:

class Rectangle

{

private:

int width;

int height;

public:

Rectangle() { width = 0; height = 0; };

Rectangle(int w, int h) { width = w; height = h;};

void setWidth(int w) { width = w;};

void setHeight(int h) { height = h;};

int getWidth() { return width;};

int getHeight() { return height; };

};

**Copy Constructors and Memory Management**

* Important to manage dynamically allocated pointer within a class
* Constructors initialize datafields that are pointer using *new*
* Destructors cleanup memory (using delete) when variable goes out of scope (or pointer to object is deleted explictly by delete)
* Copy Constructors are important when passing objects by value!

Example: (similar to CardPlayer)

class Array

{

private:

int \*p;

int size;

public:

Array(int sz);

~Array() ;

Array(const Array& a); //Copy Constructor

void put(int i, int j);

void get(int i);

};

* Notice the method declaration:
  + Array(**const** Array& a);
* The keyword **const** used in this context means that the overloaded constructor method cannot change the variable that you are passing by reference. Does it seem strange that we don’t allow the modification of a variable that we pass by reference?
* The whole purpose of the copy constructor is to create an exact copy of an existing object when it is passed to a method/function by *value*.
* Example implementation of copy constructor:

Array::Array(const Array &a)

{

int ii = 0;

size = a.size;

p = new int [a.size];

for (ii = 0; ii < size; ii++)

{

p[ii] = a.p[ii];

}

}

Important: What object does this represent in above method?

* Default copy constructor copies field-to-field
* Default copy constructor works fine in many cases
* Problem: what if object contains a pointer?

Default Copy Constructor results in pointers being shared when objects containing pointers are pass-by-value

Must define a Copy Constructor to assure that pointers are not shared and destroyed unexpectedly.

**Containment**

Up to this point, we have created classes with primitive data types:

class CardPlayer

{

private:

int capacity;

int \*hand;

int no\_cards;

public:

CardPlayer();

…..

Consider the GenString class declared below :

class GenString

{

private:

char \*string;

public:

GenString();

…..

Containment is when an object is contained within another object:

* An object can also contain another object (similar to nested structures)
* Consider the Sentence class:

class Sentence

{

private:

String Sentence;

String \*words;

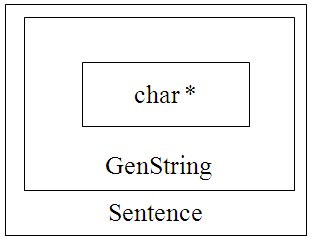
int word\_count;

public:

….

**Containment - “has-a” relationship**

* A Sentence object contains a GenString object and an int variable.
* We could say “Sentence has-a GenString”
* “GenString has-a char \*



**Order of Creation**

Order of object creation: From inside to outside.

Object data members, in the order in which

they were listed in the class declaration.

Objects that are declared as data members of

an outer object are created first before the

outer data object.

Objects are destroyed in the reverse order.

**The Sentence Class**

* char \*string within GenString is created first when a GenString object is instantiated
  + GenString(char \*); //Prototype
  + GenString string(“Joe Smith”);
* A Sentence class can be created from String class:
  + Sentence sentence(GenString string); //Prototype
  + Sentence sentence(string);

**Example main:**

int main()

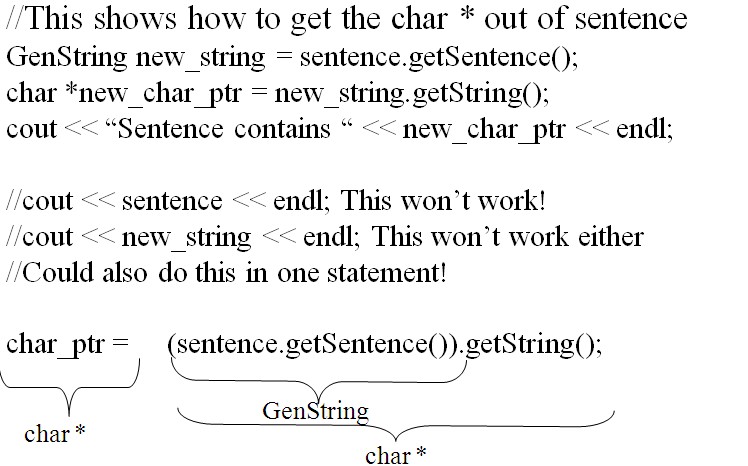
{

//This shows the initialization of a sentence object

char \*char\_ptr = “Joe Smith”;

GenString string(char\_ptr);

Sentence sentence(string);



**SentenceReverser**

Sentence could be contained in another class too!

class SentenceReverser

{

private:

Sentence sentence;

public:

Sentence getSentence();

