

CS202: Programming Systems

Week 2

Constructors & Destructor

10/2016

CS202 – What will be discussed?

- ☐ Constructors
- ☐ The **this** pointer
- ☐ Destructor
- ☐ Member Initialization
- ☐ Copy constructor
- ☐ Assignment operator

Constructors

- ❑ **Constructor** is a physical piece of code (in fact, it is a special type of method) that is used to construct and initialize objects.
- ❑ It is **automatically** invoked when a new object is created.
- ❑ There is no returned value, even a `void`.
- ❑ A class can have many constructors (overload)
- ❑ Name of the constructors must be the same as the class name.

Notes on constructors

- ❑ If no constructor is implemented, the compiler will issue a default constructor
- ❑ The default constructor:
 - No argument
 - Invoke other default constructors of data members if they are objects.
 - Doesn't initialize other data members if they are not objects.

Default constructor

- ❑ If there is at least one constructor, the default constructor will not be created by the compiler

```
class Date
{
public:
    Date(int iNewDay);
    ...
private:
    ...
};
```

```
int main()
{
    Date today; //error
    ...
    return 0;
}
```

Other constructors

- They allow users different options to create a new object

```
class Date {  
public:  
    Date();  
    Date(int, int);  
    Date(int, int, int);  
    ...  
private:  
    int iDay, iMonth, iYear;  
};
```

The **this** pointer

- Check the following lines of code, are they correct in terms of: syntax? semantics? useful?

```
Date::Date(int iDay, int iMonth, int iYear)
{
    iDay = iDay;
    iMonth = iMonth;
    iYear = iYear;
}
```

```
Date today(20, 10, 2008);
```

The **this** pointer

today

+ iDay
+ iMonth
+ iYear

tomorrow

+ iDay
+ iMonth
+ iYear

nextweek

+ iDay
+ iMonth
+ iYear

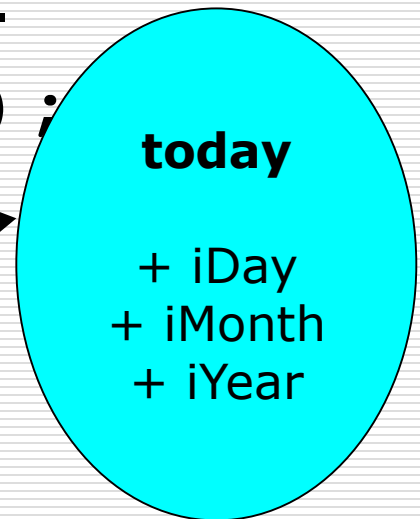
☐ How can we know **day**, **month** or **year** of which object are using?

The **this** pointer

- ❑ C++ adds an implicit function parameter - the pointer to the current object instance: **this**
- ❑ **this** is a constant pointer, you cannot modify it within a member function.

```
int month = today.getMonth();
```

```
int Date::getMonth()  
{  
    return this->iMonth;  
}
```



The **this** pointer

```
Date::Date(int iDay, int iMonth, int iYear)
{
    iDay = iDay;
    iMonth = iMonth;
    iYear = iYear;
}
```

- ☐ Syntax: correct
- ☐ Semantic: legal
- ☐ Useful: NO!!!

The code should be

```
Date::Date(int iDay, int iMonth, int iYear)
{
    this->iDay = iDay;
    this->iMonth = iMonth;
    this->iYear = iYear;
}
```

Destructor

- ❑ Invoked automatically, when the variable is removed from memory (e.g. goes out of scope).
- ❑ Each class can have at most one destructor
- ❑ The destructor name is a name of a class preceded by a tilde sign (~).
- ❑ Destructor, the same as constructor, has no return type (even void)
- ❑ Destructor frees the resources used by the object (allocated memory, file descriptors, semaphores etc.)

Example

```
class MyArray {  
public:  
    MyArray();  
    ~MyArray() {  
        n = 0;  
        delete [] pArr;  
        pArr = NULL;  
    }  
private:  
    int n;  
    int *pArr;  
};
```

Notes on destructor

- ❑ You don't need to write a destructor if your class has nothing to clean up.
- ❑ If you are using resources, for example dynamic memory allocation, and you forget to have your destructor, the program will create the **memory leaking**.

Members Initialization

❑ Distinguish between Assignment and Initialization

```
Date(int iNDay, int iNMonth, int iNYear)
{
    iDay = iNDay;
    iMonth = iNMonth;
    iYear = iNYear;
}
```

❑ This is assignment, not Initialization

Members Initialization

□ This is members initialization

```
class Date {  
public:  
    Date();  
    Date(int iNDay, int iNMonth, int iNYear)  
        : iDay(iNDay), iMonth(iNMonth), iYear(iNYear)  
    {}  
    ~Date();  
    ...  
private:  
    int iDay, iMonth, iYear;  
};
```


Mandatory Members Initialization

- ❑ Const members
- ❑ References
- ❑ Sub-objects which require arguments in constructors

```
class Test
{
private:
    Another& refA; // reference member
    const int MAX; // const member
    vector arr;
public:
    Test(Another& r) : refA(r), MAX(100), arr (MAX) {}
};
```

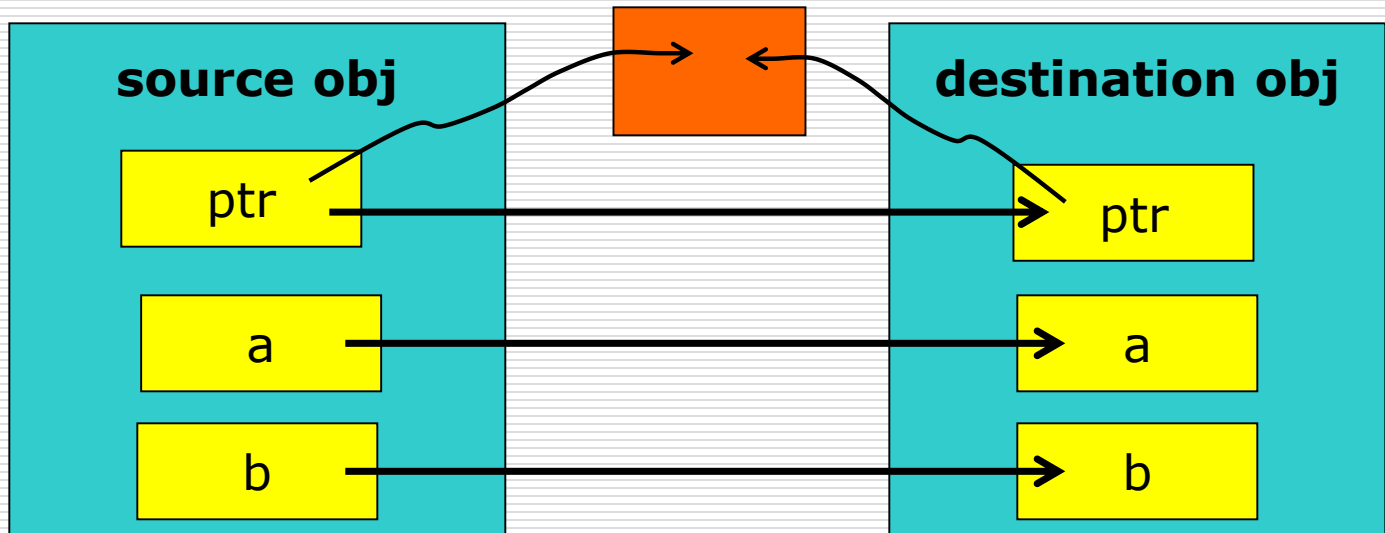
Default copy constructor

- ❑ In each class, if there is no copy constructor, a default copy constructor will be generated. It helps to create a new object of this class from another object. For example:

```
int main()
{
    Rectangle a;
    Rectangle b(a); // invoke copy constructor
    Rectangle c = a; // invoke copy constructor
}
```

Default copy constructor (cont.)

- Default copy constructor performs a bitwise copy from the source to the current object:



Copy constructor

- Due to the bitwise copy of the default constructor, it will cause a serious problem if the copying takes place when the object has a member pointer with a dynamic allocated memory.
- Pointers of the source obj and the destination obj will refer into the same memory

Copy constructor

- Depending on the members of the class to decide whether to have a copy constructor
 - When having dynamic allocated members

```
Test::Test(const Test& src)
{
    iSize = src.iSize;
    ptr = new int [iSize];
    for (int i=0; i<iSize; ++i)
        ptr[i] = src.ptr[i];
}
```

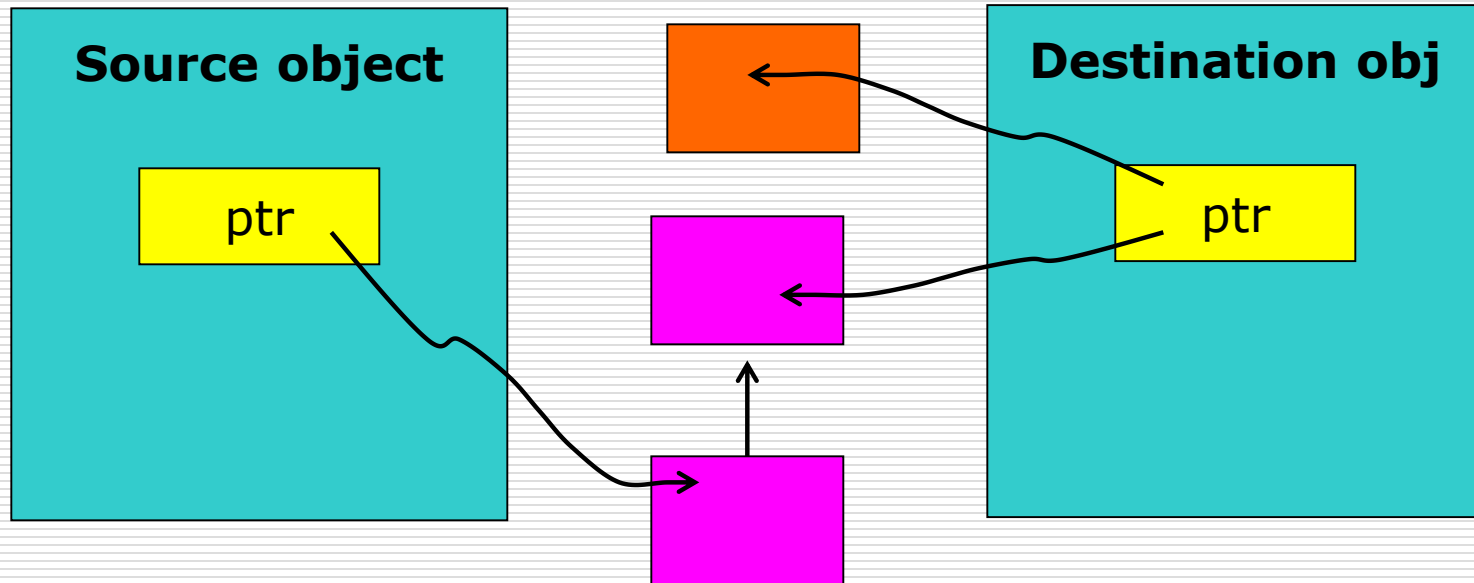
The default assignment operator

- ❑ Similar to the default copy constructor, in each class, if there is no assignment operator, a default assignment operator will be generated
- ❑ It also has a similar functionality of a default copy constructor, i.e. doing a bitwise copy from the source object to the destination object.

Assignment operator

- Thus, if there is a pointer member in the class, an assignment operator should be defined.
- Note: assignment operator is a bit different from the copy constructor:
 - Clean up the allocated memory that the pointer member is pointing to before being allocated with a new memory.
 - Remember to check for self-assignment

Assignment operator



- Clean up the memory it is pointing to
- Copy the memory to a new place

For example

```
Test& Test::operator=(const Test& src)
{
    if (this != &src)
    {
        delete [] ptr;
        iSize = src.iSize;
        ptr = new int [iSize];
        for (int i=0; i<iSize; ++i)
            ptr[i] = src.ptr[i];
    }
    return *this;
}
```

Copy constructor vs Assignment Op

Copy constructor:

- ☐ Create an object from scratch
- ☐ No need to check for self-assignment
- ☐ No need to free the resource before copying

Remember

The 3 following functions often go together:

- ☐ Copy constructor
- ☐ Assignment operator
- ☐ Destructor