The Constructors

Lecture 7
Sections 11.4 - 11.5

Robb T. Koether

Hampden-Sydney College

Wed, Feb 1, 2017

Outline

The Four Fundamental Member Functions

- The four fundamental functions
  - The default constructor
  - The copy constructor
  - The destructor
  - The assignment operator
- These four member functions are essential to the functioning of any class.
- In each case, if you fail to write your own version, the compiler will create the “automatic” version for you.

Outline

The Default Constructor

The Default Constructor

```
Type::Type();   // Prototype
Type Object;    // Usage
```

- The default constructor constructs an object for which no initial value is given.
- The default constructor should initialize the data members to neutral values that are appropriate for that type.

`Vectr` Default Constructor

Example (`Vectr` Default Constructor)

```
Vectr() : m_size(0), m_element(NULL) {}
```

Purposes of the Default Constructor

The Default Constructor

```
Vectr v;
Vectr* ptr = new Vectr[10];
Vectr v_arr[10] = {Vectr(4, 123), Vectr(8, 567)};
```

- The default constructor is used when
  - An object is created with no initial value specified.
  - The **new** operator is used to create an array.
  - A static array is partially initialized.

Outline

The Automatic Default Constructor

- The automatic default constructor is provided automatically if we write no constructor.
- It
  - Allocates memory for the data members.
  - Invokes each data member's own default constructor.
- However, if we write any constructor, then the automatic default constructor is *not* provided.
- In that case, we must write the default constructor, if we want one.

Outline

The Copy Constructor

The Copy Constructor

```
Type::Type(const Type&);   // Prototype
Type ObjectA = ObjectB;    // Usage 1
Type ObjectA(ObjectB);     // Usage 2
```

- The copy constructor constructs an object which will be a copy of an existing object.

Vectr Copy Constructor

Example (Vectr Copy Constructor)

```
Vectr(const Vectr& v)
{
    m_size = v.m_size;

    if (m_size == 0)
        m_element = NULL;
    else
        m_element = new double[m_size];

    for (int i = 0; i < m_size; i++)
        m_element[i] = v.m_element[i];
    return;
}
```

Purposes of the Copy Constructor

The Copy Constructor

```
Vectr f(Vectr v);
int main()
{
    Vectr v;
    Vectr u = v;
    Vectr w(u);
    u = f(v);
        ⋮
}
```

- The copy constructor is used when
  - An object is created and initialized to the value of an existing object of the same type.
  - A local copy of a value parameter is created during a function call.
  - A function returns a value.

Point of Style

The Copy Constructor

```
Vectr u = v;                    // Good style
Vectr u(v);                     // Good style
Vectr u;                        // Poor-
u = v;                          //      style
Vectr u = Vectr(3, 123);        // Poor style
```

- The first and second use the copy constructor.
- The third and fourth lines use the default constructor followed by the assignment operator.
- The fifth uses another constructor followed by the copy constructor.

makeCopy

Example (makeCopy)

```
void makeCopy(const Vectr& v)
{
    m_size = v.m_size;

    if (m_size == 0)
        m_element = NULL;
    else
        m_element = new double[m_size];

    for (int i = 0; i < m_size; i++)
        m_element[i] = v.m_element[i];
    return;
}
```

- A handy technique is to write a function `makeCopy()` and simply call on it to do the work of the copy constructor.

`Vectr` Copy Constructor

Example (`Vectr` Copy Constructor)

```
Vectr(const Vectr& v)
{
    makeCopy(v);
    return;
}
```

Points of Style

The Copy Constructor

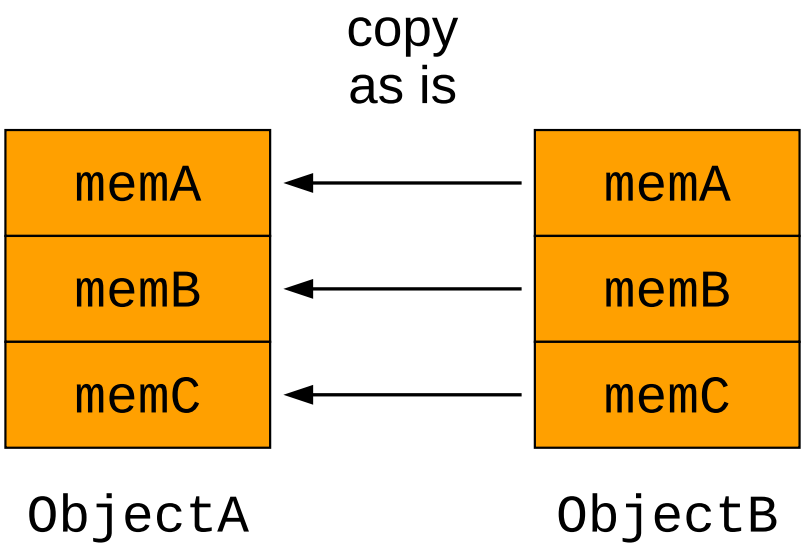
```
Type::Type(const Type& obj)
{
    makeCopy(obj);
    return;
}

void Type::makeCopy(const Type& obj)
{
    //  Make a copy
}
```

Outline

The Automatic Copy Constructor

- The automatic copy constructor
  - Allocates memory for the data members.
  - Invokes each data member's copy constructor to copy values from the existing object.



The Automatic Copy Constructor

- This is called a shallow copy.
- Pointers get copied with no change in value.
- Therefore, the pointer in the new object will point to the very same memory as the pointer in the old object.
- Generally, this is not good. Instead, we want a deep copy.
- What would happen in the `Vectr` class if we made a shallow copy of a vector?

Outline

Constructors and the `new` Operator

The `new` Operator and Constructors

```
Vectr* ptr;
ptr = new Vectr;
ptr = new Vectr(v);
ptr = new Vectr(10);
ptr = new Vectr(10, 123);
ptr = new Vectr[10];
```

- The **`new`** operator is designed to work in conjunction with the constructors.

Outline

Assignment

Assignment

- Read Sections 11.4 - 11.5.