

A decorative vertical bar on the left side of the slide. It consists of a dark teal background with a white vertical stripe. To the right of the stripe are several orange circles of varying sizes, and a thin orange vertical line runs parallel to the stripe.

OBJECT ORIENTED PROGRAMMING USING C++

Constructors & Destructors Review

What is a constructor?

- It is a member function which initializes a class.
- A constructor has:
 - (i) the same name as the class itself
 - (ii) no return type

```
class rectangle {  
private:  
    float height;  
    float width;  
    int xpos;  
    int ypos;  
public:  
    rectangle(float, float); // constructor  
    void draw(); // draw member function  
    void posn(int, int); // position member function  
    void move(int, int); // move member function  
};
```

```
rectangle::rectangle(float h, float w)  
{  
    height = h;  
    width = w;  
    xpos = 0;  
    ypos = 0;  
}
```

Comments on constructors

- A constructor is called automatically whenever a new instance of a class is created.
- You must supply the arguments to the constructor when a new instance is created.
- If you do not specify a constructor, the compiler generates a default constructor for you (expects no parameters and has an empty body).

Comments on constructors (cont.)

```
void main()
{
    rectangle rc(3.0, 2.0);

    rc.posn(100, 100);
    rc.draw();
    rc.move(50, 50);
    rc.draw();
}
```

- *Warning:* attempting to initialize a data member of a class explicitly in the class definition is a syntax error.

Overloading constructors

- You can have more than one constructor in a class, as long as each has a different list of arguments.

```
class rectangle {
private:
    float height;
    float width;
    int xpos;
    int ypos;
public:
    rectangle(float, float); // constructor
    rectangle();             // another constructor
    void draw();             // draw member function
    void posn(int, int);     // position member function
    void move(int, int);     // move member function
};
```

Overloading constructors (cont.)

```
rectangle::rectangle()
{
    height = 10;
    width = 10;
    xpos = 0;
    ypos = 0;
}
```

```
void main()
{
    rectangle rc1(3.0, 2.0);
    rectangle rc2();

    rc1.draw();
    rc2.draw();
}
```


Composition: objects as members of classes

- A class may have objects of other classes as members.

```
class properties {  
    private:  
        int color;  
        int line;  
    public:  
        properties(int, int); // constructor  
};
```

```
properties::properties(int c, int l)  
{  
    color = c;  
    line = l;  
}
```

Composition: objects as members of classes (cont.)

```
class rectangle {  
private:  
    float height;  
    float width;  
    int xpos;  
    int ypos;  
    properties pr;          // another object  
public:  
    rectangle(float, float, int, int ); // constructor  
    void draw();           // draw member function  
    void posn(int, int);   // position member function  
    void move(int, int);   // move member function  
};
```

Composition: objects as members of classes (cont.)

```
rectangle::rectangle(float h, float w, int c, int l):pr(c, l)
{
    height = h;
    width = w;
    xpos = 0;
    ypos = 0;
};
```

```
void main()
{
    rectangle rc(3.0, 2.0, 1, 3);

    C++ statements;
}
```

What is a destructor?

- It is a member function which deletes an object.
- A destructor function is called automatically when the object goes out of scope:
 - (1) the function ends
 - (2) the program ends
 - (3) a block containing temporary variables ends
 - (4) a *delete* operator is called
- A destructor has:
 - (i) the same name as the class but is preceded by a tilde (~)
 - (ii) no arguments and return no values

```
class string {
private:
    char *s;
    int size;
public:
    string(char *); // constructor
    ~string();      // destructor
};
```

```
string::string(char *c)
{
    size = strlen(c);
    s = new char[size+1];
    strcpy(s,c);
}
```

```
string::~~string()
{
    delete []s;
}
```

Comments on destructors

- If you do not specify a destructor, the compiler generates a default destructor for you.
- When a class contains a pointer to memory you allocate, **it is your responsibility** to release the memory before the class instance is destroyed.

What is a copy constructor?

- It is a member function which initializes an object using another object of the same class.
- A copy constructor has the following general function prototype:

class_name (const class_name&);

```
class rectangle {
private:
    float height;
    float width;
    int xpos;
    int ypos;
public:
    rectangle(float, float);           // constructor
    rectangle(const rectangle&); // copy constructor
    void draw();                       // draw member function
    void posn(int, int);               // position member function
    void move(int, int);               // move member function
};
```



```
rectangle::rectangle(const rectangle& old_rc)
{
    height = old_rc.height;
    width = old_rc.width;
    xpos = old_rc.xpos;
    ypos = old_rc.ypos;
}
```

```
void main()
{
    rectangle rc1(3.0, 2.0);           // use constructor
    rectangle rc2(rc1);               // use copy constructor
    rectangle rc3 = rc1;             // alternative syntax for
                                    // copy constructor

    C++ statements;
}
```

Defining copy constructors is very important

- In the absence of a copy constructor, the C++ compiler builds a default copy constructor for each class which is doing a memberwise copy between objects.
- Default copy constructors work fine unless the class contains pointer data members ...
why???

```
#include <iostream.h>
#include <string.h>

class string {
private:
    char *s;
    int size;
public:
    string(char *); // constructor
    ~string();      // destructor
    void print();
    void copy(char *);
};

void string::print()
{
    cout << s << endl;
}
```

```
void string::copy(char *c)
{
    strcpy(s, c);
}
```

```
void main()
{
    string str1("George");
    string str2 = str1;    // default copy constructor

    str1.print();    // what is printed ?
    str2.print();

    str2.copy("Mary");

    str1.print();    // what is printed now ?
    str2.print();
}
```

Defining a copy constructor for the above example:

```
class string {  
private:  
    char *s;  
    int size;  
public:  
    string(char *);           // constructor  
    ~string();               // destructor  
    string(const string&);   // copy constructor  
    void print();  
    void copy(char *);  
};
```

```
string::string(const string& old_str)
{
    size = old_str.size;
    s = new char[size+1];
    strcpy(s,old_str.s);
}
```

```
void main()
{
    string str1("George");
    string str2 = str1;

    str1.print(); // what is printed ?
    str2.print();

    str2.copy("Mary");

    str1.print(); // what is printed now ?
    str2.print();

}
```

Note: same results can be obtained by overloading the assignment operator.