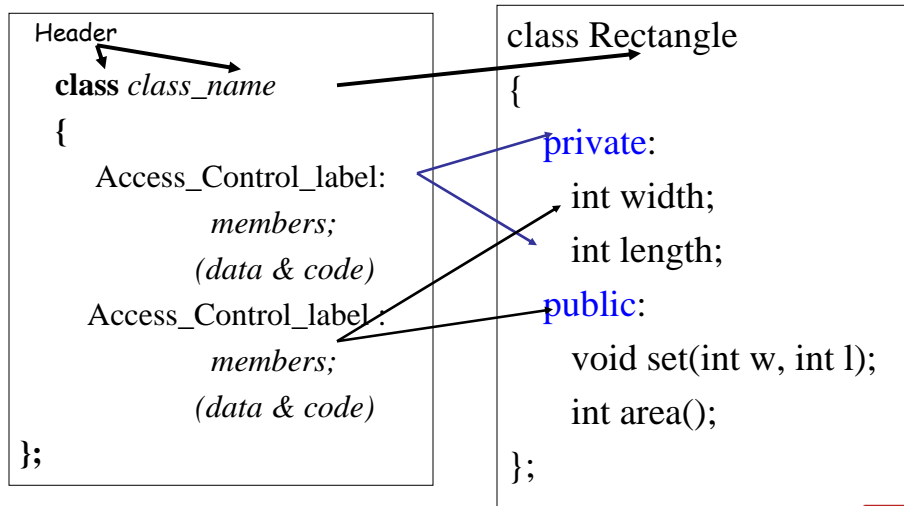


Review: Defining a class



```

class Date {
    int d, m, y;
    static Date default_date;
public:
    Date ( int dd=0, int mm=0, int yy=0);
    static void set_default (int dd, int mm, int yy);
    int day () const { return d; }
    int year () const ;
};
    
```

Default values

```
Date my_date = Date ();
```

What is the value contained in my_date ??

Constructor

```

Date::Date ( int dd, int mm, int yy ) {
    d = dd ? dd : default_date.d;
    m = mm ? mm : default_date.m;
    y = yy ? yy : default_date.y;
}
    
```

Qualify with class name to define outside

```

int Date::year () const {
    y++; // invalid
    return y;
}
    
```

Static keyword can be skipped!

```

Date Date::default_date(16, 1, 2006);
void Date::set_default ( int d, int m, int y ) {
    default_date = Date (d, m, y);
}
    
```

:: - Scope Operator

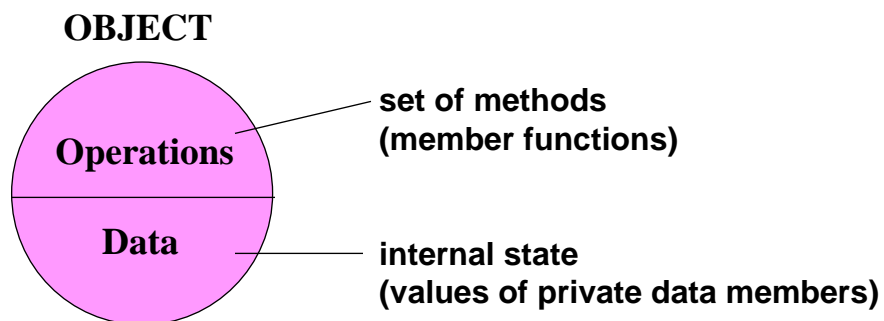


Objects

- Objects: variables or instances of a class that is declared in a program
- Declaration of an Object
- Initiation of Objects
- Constructors & Destructors
- Working with Multiple Files



What is an object?

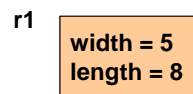


Declaration of an Object

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
}
```

r1 is statically allocated

```
main()
{
    Rectangle r1;
    r1.set(5, 8);
}
```

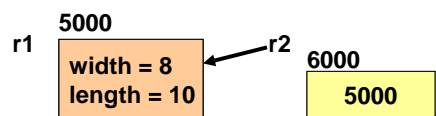


Declaration of an Object

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
}
```

r2 is a pointer to a Rectangle object

```
main()
{
    Rectangle r1;
    r1.set(5, 8); //dot notation
    Rectangle *r2;
    r2 = &r1;
    r2->set(8,10); //arrow notation
}
```



Declaration of an Object

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
}
```

r3 is dynamically allocated

```
main()
{
    Rectangle *r3;
    r3 = new Rectangle();

    r3->set(80,100); //arrow notation

    delete r3;
    r3 = NULL;
}
```

r3
6000
NULL

Copyright 2006, The Ohio State University

Object Initialization

```
#include <iostream.h>

class circle
{
    public:
        double radius;
};
```

1. By Assignment

- Only work for public data members
- No control over the operations on data members

```
int main()
{
    circle c1; // Declare an instance of the class circle
    c1.radius = 5; // Initialize by assignment

    circle c2 = {-10}; // Initialize by aggregation, not valid for a circle
}
```

Copyright 2006, The Ohio State University

8

University

Object Initialization

```
#include <iostream.h>

class circle
{
private:
    double radius;

public:
    void set (double r)
        {radius = r;}
    double get_r ()
        {return radius;}
};
```

2. By Public Member Functions

- Accessor
- Implementor

```
int main(void) {
    circle c;           // an object of circle class
    c.set(5.0);         // initialize an object with a public member function
    cout << "The radius of circle c is " << c.get_r() << endl;
    // access a private data member with an accessor
}
```

Declaration of an Object

```
class Rectangle
{
private:
    int width;
    int length;
public:
    void set(int w, int l);
    int area();
}
```

r2 is a pointer to a Rectangle object

```
main()
{
    Rectangle r1;
    r1.set(5, 8);    //dot notation

    Rectangle *r2;
    r2 = &r1;
    r2->set(8,10);  //arrow notation
}
```

r1 and r2 are both initialized by public member function set

Object Initialization

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        Rectangle();
        Rectangle(const Rectangle &r);
        Rectangle(int w, int l);
        void set(int w, int l);
        int area();
}
```

3. By Constructor

- Default constructor
- Copy constructor
- Constructor with parameters

They are publicly accessible
Have the same name as the class
There is no return type
Are used to initialize class data members
They have different signatures

Object Initialization

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
}
```

When a class is declared with no constructors, the compiler automatically assumes default constructor and **copy** constructor for it.

- Default constructor

```
Rectangle :: Rectangle() { };
```

- Copy constructor

```
Rectangle :: Rectangle (const Rectangle &
r)
{
    width = r.width; length = r.length;
};
```

Object Initialization

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
}
```

- Initialize with **default** constructor

```
Rectangle r1;
Rectangle *r3 = new Rectangle();
```

- Initialize with **copy** constructor

```
Rectangle r4;
r4.set(60,80);

Rectangle r5 = Rectangle(r4);
Rectangle *r6 = new Rectangle(r4);
```



Object Initialization

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        Rectangle(int w, int l)
            { width =w; length=l;}
        void set(int w, int l);
        int area();
}
```

If any constructor with any number of parameters is declared, no **default** constructor will exist, unless you define it.

```
Rectangle r4; // error
```

- Initialize with **constructor**

```
Rectangle r5(60,80);
Rectangle *r6 = new Rectangle(60,80);
```



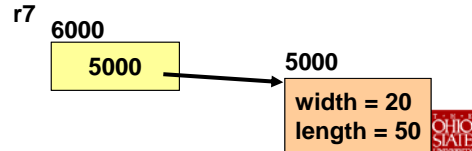
Object Initialization

```
class Rectangle
{
    private:
        int width;
        int length;
    public:
        Rectangle();
        Rectangle(int w, int l);
        void set(int w, int l);
        int area();
}
```

Write your own constructors

```
Rectangle :: Rectangle()
{
    width = 20;
    length = 50;
};
```

```
Rectangle *r7 = new Rectangle();
```



Copyright 2006, The Ohio State University

Object Initialization

```
class Account
{
    private:
        char *name;
        double balance;
        unsigned int id; //unique
    public:
        Account();
        Account(const Account &a);
        Account(const char *person);
}
```

```
Account :: Account()
{
    name = NULL; balance = 0.0;
    id = get_unique_id();
};
```

With constructors, we have more control over the data members

```
Account :: Account(const Account &a)
{
    name = new char[strlen(a.name)+1];
    strcpy (name, a.name);
    balance = a.balance;
    id = get_unique_id();
};
```

```
Account :: Account(const char *person)
{
    name = new char[strlen(person)+1];
    strcpy (name, person);
    balance = 0.0;
    id = get_unique_id();
};
```

ght 2

So far, ...

- An object can be initialized by a class constructor
 - default constructor
 - copy constructor
 - constructor with parameters
- Resources are allocated when an object is initialized
- Resources should be revoked when an object is about to end its lifetime



Cleanup of An Object

```
class Account
{
private:
    char *name;
    double balance;
    unsigned int id; //unique
public:
    Account();
    Account(const Account &a);
    Account(const char *person);
    ~Account();
}
```

Destructor

```
Account :: ~Account()
{
    delete[] name;
    release_id (id);
}
```

- Its name is the class name preceded by a ~ (tilde)
- It has no argument
- It is used to release dynamically allocated memory and to perform other "cleanup" activities
- It is executed automatically when the object goes out of scope

Putting Them Together

```
class Str
{
    char *pData;
    int nLength;
public:
    //constructors
    Str();
    Str(char *s);
    Str(const Str &str);

    //accessors
    char* get_Data();
    int get_Len();

    //destructor
    ~Str();
};
```

```
Str :: Str() {
    pData = new char[1];
    *pData = '\0';
    nLength = 0;
};
```

```
Str :: Str(char *s) {
    pData = new char[strlen(s)+1];
    strcpy(pData, s);
    nLength = strlen(s);
};
```

```
Str :: Str(const Str &str) {
    int n = str.nLength;
    pData = new char[n+1];
    nLength = n;
    strcpy(pData, str.pData);
```

Copyright 2006, The Ohio State University



Putting Them Together

```
class Str
{
    char *pData;
    int nLength;
public:
    //constructors
    Str();
    Str(char *s);
    Str(const Str &str);

    //accessors
    char* get_Data();
    int get_Len();

    //destructor
    ~Str();
};
```

```
char* Str :: get_Data()
{
    return pData;
};
```

```
int Str :: get_Len()
{
    return nLength;
};
```

```
Str :: ~Str()
{
    delete[] pData;
};
```

Copyright 2006, The Ohio State University

20



Putting Them Together

```
class Str
{
    char *pData;
    int nLength;
public:
    //constructors
    Str();
    Str(char *s);
    Str(const Str &str);

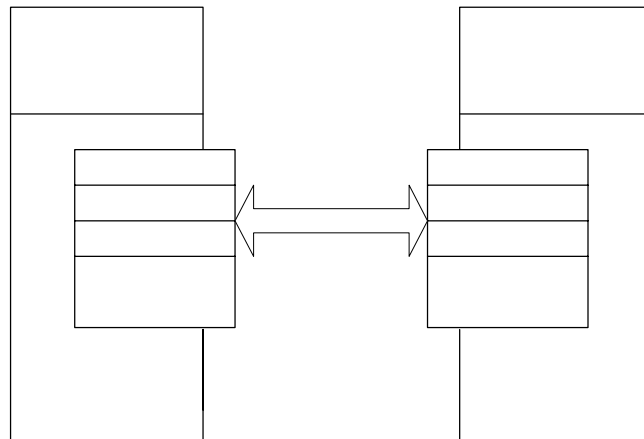
    //accessors
    char* get_Data();
    int get_Len();

    //destructor
    ~Str();
};
```

```
int main()
{
    int x=3;
    Str *pStr1 = new Str("Joe");
    //initialize from char*
    Str *pStr2 = new Str();
    //initialize with constructor
}
```



Interacting Objects



Take Home Message

- There are different types of member functions in the definition of a class
 - **Accessor**
 - `int CStr :: get_length();`
 - **implementor/worker**
 - `void Rectangle :: set(int, int);`
 - **helper**
 - `void Date :: errmsg(const char* msg);`
 - **constructor**
 - `Account :: Account();`
 - `Account :: Account(const Account& a);`
 - `Account :: Account(const char *person);`
 - **destructor**
 - `Account :: ~Account();`

Copyright 2006, The Ohio State University

23



Working with Multiple Files

- To improve the readability, maintainability and reusability, codes are organized into modules.
- When working with complicated codes,
 - A set of `.cpp` and `.h` files for each class groups
 - `.h` file contains the prototype of the class
 - `.cpp` contains the definition/implementation of the class
 - A `.cpp` file containing `main()` function, should include all the corresponding `.h` files where the functions used in `.cpp` file are defined

Copyright 2006, The Ohio State University

24



Example : time.h

```
// SPECIFICATION FILE           ( time .h )
// Specifies the data members and
// members function prototypes.

#ifndef TIME_H
#define TIME_H

class Time
{
public:
    . . .

private:
    . . .
};

#endif
```



Example : time.cpp

```
// IMPLEMENTATION FILE         ( time.cpp )
// Implements the member functions of class Time
#include "time.h"           // also must appear in client code
#include <iostream.h>
. . .

bool Time :: Equal ( Time otherTime ) const
//      Function value == true,  if this time equals otherTime
//                        == false , otherwise
{
    return ( (hrs == otherTime.hrs) && (mins == otherTime.mins)
              && (secs == otherTime.secs) );
}

. . .
```



Example : main.cpp

```
// Client Code      ( main.cpp )
#include "time.h"

// other functions, if any

int main()
{
    .....
}
```

Compile and Run

```
g++ -o main main.cpp time.cpp
```



Separate Compilation and Linking of Files

