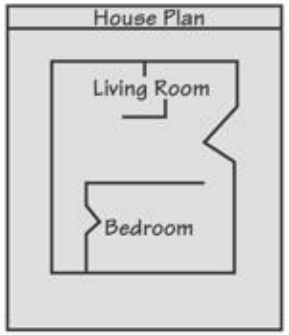
Object-oriented terminology

- class: An entity that has **attributes** (data members or fields) and **behaviors** (function members or methods)

- object: an instance of a `class`

- A class is like a blueprint and objects are like houses built from the blueprint.
- A class is a **user-defined type**.
- A class may also be viewed as an **abstraction over objects**

Blueprint that describes a house.



Instances of the house described by the blueprint.

Classes in C++

- In C++, a `class` is similar to a `struct`. They both may have data members (attributes) and function members (methods)
- However, by **default**, data members in a class are **not accessible** outside the class.

- **Objects** are created from a `class`, just like they are created from `struct`.

- Format:

```
class ClassName
{
        declaration;
        declaration;
};
```

Object-oriented concepts

- Data encapsulation:
  - Merging of attributes and behaviors
  - Defining not only data members, but also methods for accessing them

- Data hiding:
  - Restricting access to certain members of an object → use access specifiers

Example – how do we encapsulate here?

```
#include <iostream>
#include <string>
using namespace std;

struct Rectangle {
    string name;
    int width;
    int height;

};

void printRectangle(const Rectangle &rec) {
    cout << rec.name << " width: " << rec.width;
    cout << ", height: " << rec.height << endl;
}

int main() {
    Rectangle myrec;
    myrec.name = "Rectangle_1";
    myrec.width = 4;
    myrec.height = 2;
    printRectangle(myrec);
}
```

```
Rectangle_1 width: 4, height: 2
```

Example - encapsulation

```
#include <iostream>
#include <string>
using namespace std;

struct Rectangle {
    string name;
    int width;
    int height;

    void printRectangle() const{
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }
};


int main() {
    Rectangle myrec;
    myrec.name = "Rectangle 1";
    myrec.width = 4;
    myrec.height = 2;
    myrec.printRectangle();
}
```

Rectangle_1 width: 4, height: 2

Merging of attributes (struct fields) and methods → **encapsulation**

`const` appearing after the parentheses in a member function declaration specifies that the function will not change any data in its object.

Access specifiers

- Used to control access to members of struct or class
- `public:` can be accessed by functions outside of the class
- `private:` can only be called by, or accessed by function members of the class
- public interface:
  - Members of an object (data or methods) that are accessible outside of the object.
  - If you are using a class then, for this course, **all data members *should* be private**.
    - Stroustrup takes this position.

Access specifiers cont.

- Can be listed in any order in a class
- Can appear multiple times in a class
- If not specified
  - class: default is `private`
  - struct: default is `public`

OOP concepts – data hiding

```
#include <iostream>
#include <string>
using namespace std;

struct Rectangle {

    void printRectangle() const{
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }

private:
    string name;
    int width;
    int height;
};


int main() {
    Rectangle myrec;
    myrec.name = "Rectangle_1";
    myrec.width = 4;
    myrec.height = 2;
    myrec.printRectangle();
}
```

Access specifiers used to achieve data hiding

Compilation error – cannot access private data members from outside the struct

OOP concepts – data hiding cont.

```cpp
#include <iostream>
#include <string>
using namespace std;

class Rectangle {

    void printRectangle() const{
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }

private:
    string name;
    int width;
    int height;
};


int main() {
    Rectangle myrec;
    myrec.name = "Rectangle_1";
    myrec.width = 4;
    myrec.height = 2;
    myrec.printRectangle();
}
```

Using class instead of struct

More compilation errors:

1) Cannot access private data members from outside the struct
2) printRectangle is also private!
   In classes, everything is private by default!

OOP concepts – data hiding cont.

```
#include <iostream>
#include <string>
using namespace std;

class Rectangle {
public:
    // setters (mutators)
    void setname(const string& iname) { name = iname; }
    void setwidth(int iwidth) { width = iwidth; }
    void setheight(int iheight) { height = iheight; }

    void printRectangle() const {
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }
private:
    string name;
    int width;
    int height;
};

int main() {
    Rectangle myrec;
    myrec.setname("Rectangle_1");
    myrec.setwidth(4);
    myrec.setheight(2);
    myrec.printRectangle();
}
```

Rectangle_1 width: 4, height: 2

All seems good now….

1) But there's a time between object creation and setting the values → particularly important if the object is instantiated as a global variable

2) Class user may not be fully aware of internals of class and may prefer for some values to be initialized for him/her by default

Constructors

- Member function that is automatically called when an object is created
- Purpose is to construct an object
- Constructor function name is the same as the class name
- Has no return type

Passing arguments to constructors – cont.

- You can pass arguments to the constructor when you create an object:

```
Rectangle r(“Rectangle_1”, 4, 2);
```

Constructors

```
#include <iostream>
#include <string>
using namespace std;

class Rectangle {
public:
    Rectangle(const string& iname, int iwidth, int iheight){
        name = iname;
        width = iwidth;
        height = iheight;
    }
    void printRectangle() const {
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }
private:
    string name;
    int width;
    int height;
};

int main() {
    Rectangle myrec("Rectangle_1", 4,2);
    myrec.printRectangle();
}
```

Rectangle_1 width: 4, height: 2

All is good now ..

Default constructors

- A default constructor is a constructor that takes no arguments.
  - Not to be confused with a “built-in” constructor; a constructor that’s provided by C++.

- If you write a class with no constructor at all, C++ will write a default constructor for you (i.e. a built-in default constructor), one that does nothing.

- Conversely, if you write any constructor for your class, the default constructor is not provided and you must declare one if you need it.

- A simple instantiation of a class (with no arguments) calls the default constructor:

```
Rectangle r;
```

Classes with no default constructors

- When all of a class's constructors require arguments, then the class has NO default constructor.

- When this is the case, you must pass the required arguments to the constructor when creating an object.

Initialization of data members

- **Non-primitive fields** are always initialized in the **initialization list**,
  - whether you write the initialization list or not
  - if you don't specify anything for them, they use their type's *default* constructor
  - So, to avoid doing work only to undo it, you should always use the initialization list for non-primitive fields
- **Primitive types** are **not** initialized by default (if objects are declared as local variables)
  - And so it doesn't matter if they are set in the initialization list or the body

Constructors – initialization list

```
#include <iostream>
#include <string>
using namespace std;

class Rectangle {
public:
    Rectangle(const string& iname, int iwidth, int
iheight) : name(iname),width(iwidth), height(iheight)
{    }
    void printRectangle() const {
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }
private:
    string name;
    int width;
    int height;
};

int main() {
    Rectangle myrec("Rectangle_1", 4,2);
    myrec.printRectangle();
}
```

Rectangle_1 width: 4, height: 2

Using an initialization list..

Constructors – built-in default constructor??

```cpp
#include <iostream>
#include <string>
using namespace std;

class Rectangle {
public:
    Rectangle(const string& iname, int iwidth, int iheight) :
name(iname), width(iwidth), height(iheight) {    }
    void printRectangle() const {
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }
private:
    string name;
    int width;
    int height;
};

int main() {
    Rectangle myrec("Rectangle_1", 4, 2);
    Rectangle myrec2;
    myrec.printRectangle();
}
```

```
Build started...
1>------ Build started: Project: Lect_01_B,
Configuration: Release x64 ------
1>Lect_04_classes.cpp
1>C:\Dropbox\CS2124_OOP_24S\Lect_04_classes.c
pp(20,15): error C2512: 'Rectangle': no appropriate
default constructor available
1>C:\Dropbox\CS2124_OOP_24S\Lect_04_classes.c
pp(5,7): message : see declaration of 'Rectangle'
1>Done building project "cs2124.vcxproj" -- FAILED.
========== Build: 0 succeeded, 1 failed, 0 up-to-
date, 0 skipped ==========
```

- When we define a constructor of our own → **The built-in default constructor is gone**.
  - Cannot instantiate an object without passing init parametrs.

Constructors - cont.

```
#include <iostream>
#include <string>
using namespace std;

class Rectangle {
public:
    Rectangle(): name("Unnamed"), width(1), height(1) {
    }
    void printRectangle() const {
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }
private:
    string name;
    int width;
    int height;
};

int main() {
    Rectangle myrec;
    myrec.printRectangle();
}
```

Unnamed width: 1, height: 1

Using a default constructor

Inspectors and mutators

- Inspector (getters): function that retrieves a value from a private member variable.
  - Inspectors do not change an object's data, **so they should be marked `const`.**

- Mutator (setters): a member function that stores a value in a private member variable, or changes its value in some way
  - e.g: setwidth(int), setheight(int), etc.

Inspectors & member function definition

```
#include <iostream>
#include <string>
using namespace std;

class Rectangle {
public:
    Rectangle(const string& iname, int iwidth, int iheight) :
name(iname),width(iwidth), height(iheight) {
    }
    Rectangle(): name("Unnamed"), width(1), height(1) {}
    void printRectangle() const {
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }
    // inspector
    const string& getname() const;
private:
    string name;
    int width;
    int height;
};

const string& Rectangle::getname() const{
    return name;
}
int main() {
    Rectangle myrec("Rectangle_1",4,2);
    cout << myrec.getname() << endl;
}
```

Rectangle_1

Always use const with inspectors (getters)

Function definition must include the class name, the tag.

Friend functions

```
#include <iostream>
#include <string>
using namespace std;

class Rectangle {
public:
    Rectangle(const string& iname, int iwidth, int iheight) :
name(iname),width(iwidth), height(iheight) {
    }
    Rectangle(): name("Unnamed"), width(1), height(1) {}
    void printRectangle() const {
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }
    friend const string& getname(const Rectangle&);
private:
    string name;
    int width;
    int height;
};
const string& getname(const Rectangle& irec) {
    return irec.name;
}
int main() {
    Rectangle myrec;
    cout << getname(myrec) << endl;
}
```

Unnamed

- A friend function can access an object's private data and functions.
- A friend function is **NOT** a member function.
  - Thus requires no class tag in its definition.

Overloading the output operator

```
#include <iostream>
#include <string>
using namespace std;

class Rectangle {
public:
    Rectangle(const string& iname, int iwidth, int iheight) :
name(iname),width(iwidth), height(iheight) {
    }
    Rectangle(): name("Unnamed"), width(1), height(1) {}
    void printRectangle() const {
        cout << name << " width: " << width;
        cout << ", height: " << height << endl;
    }
    // inspector
    friend ostream& operator<<(ostream& os, const Rectangle& irec);
private:
    string name;
    int width;
    int height;
};

ostream& operator<<(ostream& os, const Rectangle& rec) {
    os << rec.name << " width: " << rec.width;
    os << ", height: " << rec.height;
    return os;
}
int main() {
    Rectangle myrec("Rectangle_1",4,2);
    cout << myrec << endl;
}
```

```
Rectangle_1 width: 4, height: 2
```

Operator << is overloaded

It needs to be a friend function to access the objects private data