ECE 220 Computer Systems & Programming

Intro to C++

ECE ILLINOIS

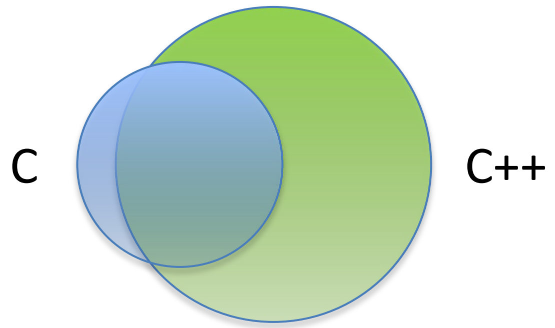
ILLINOIS

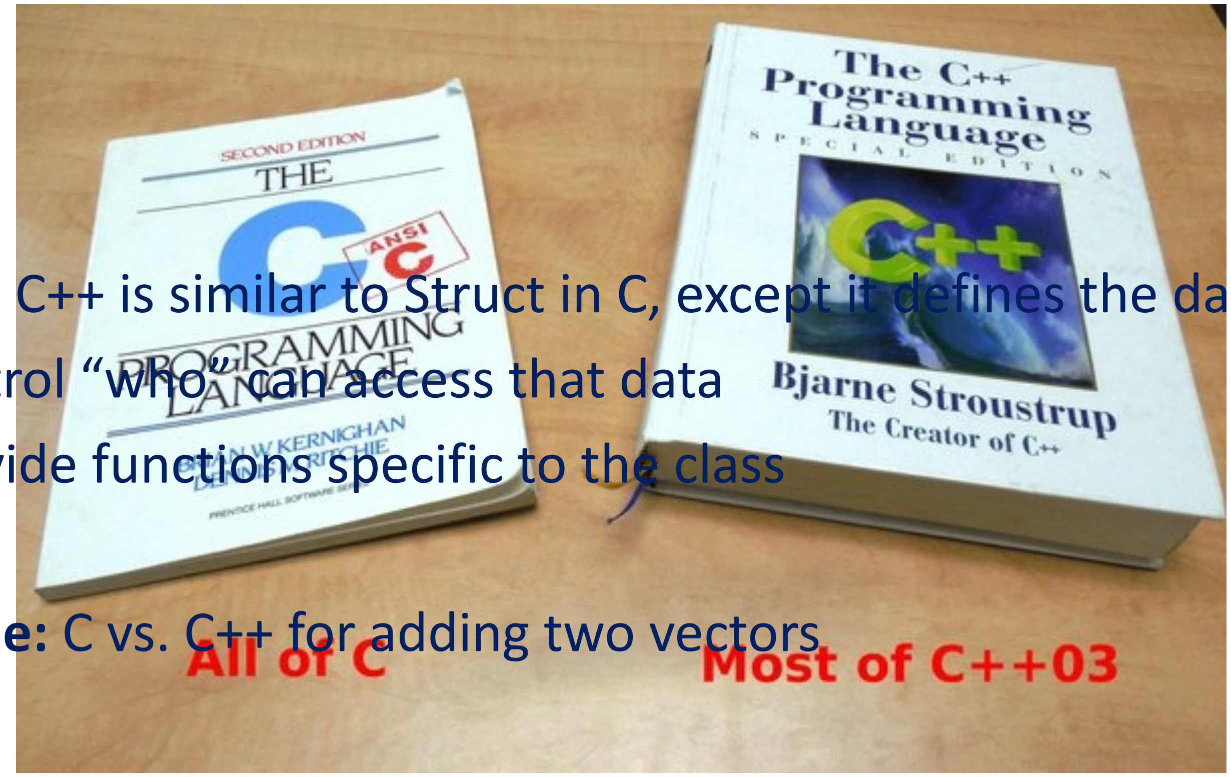
C++ - Class & Encapsulation

- Created in 1979 by Bjarne Stroustrup at Bell Labs, as an extension to C
- It's an **object oriented** language

  OOP Concepts:

  Encapsulation, Inheritance, Polymorphism, Abstraction





Class in C++ is similar to Struct in C, except it defines the data structure **AND**

- control "who" can access that data
- provide functions specific to the class

Example: C vs. C++ for adding two vectors

Concepts Related to Class

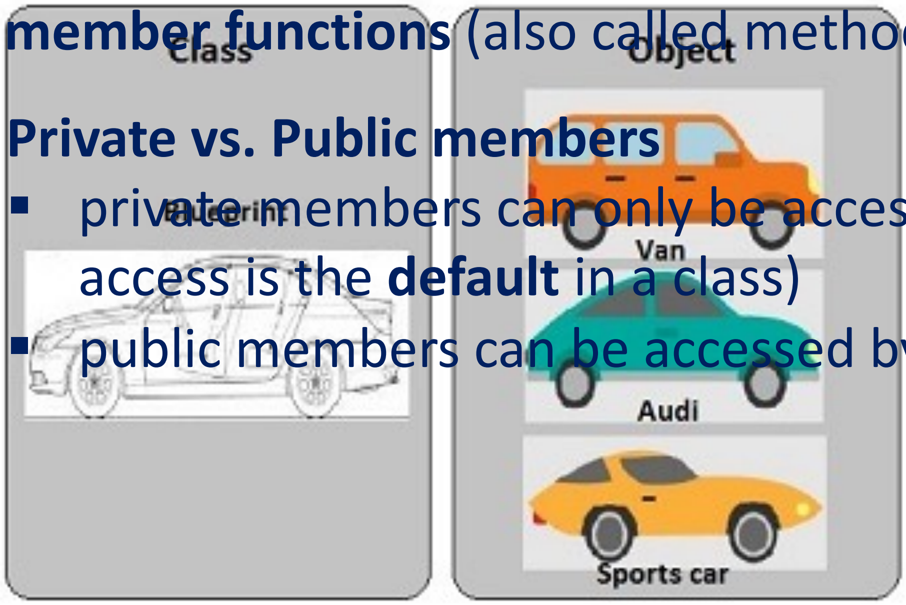
An **object** is an instance of the class

- shares the same functions with other objects of the same class
- but each object has its own copy of the data

member functions (also called methods) - functions that are part of a class

Private vs. Public members

- private members can only be accessed by member functions (private access is the **default** in a class)
- public members can be accessed by anyone



ction that *creates* (initiates) a new object

- Destructor – a special member function that *deletes* an object (when it goes outside of scope)

Basic Input / Output

cin – standard input stream
cout – standard output stream

namespace –
"using namespace" directive tells compiler the subsequent code is using names in a specific namespace (otherwise you need to use std::identifier)

Example:
```
#include <iostream>
using namespace std;
int main(){
      char name[20];
      cout << "Enter your name: ";
      cin >> name; //cin.getline(name, sizeof(name));
      cout << "Your name is: " << name << endl;
}
```

Exercise – Write Constructors

```
class Rectangle(
      int width, height;
   public:
      Rectangle();
      Rectangle(int, int);
      int area() {return width*height;}
};
Rectangle::Rectangle(){
//set both width and height to 0

}
Rectangle::Rectangle(int a, int b){
//set width to a and height to b



}
```

Exercise – Access Member in a Class

```
int main(){
      Rectangle rect1(3,4);
      Rectangle rect2;

      //print rect1's area


      //print rect2's area


      return 0;
}
```

What is the area of rect1? How about rect2?

Dynamic Memory Allocation

new – operator to *allocate* memory (similar to *malloc* in C)

delete – operator to *deallocate* memory (similar to *free* in C)

Use **delete []** whenever you allocated as an array

Example:

```
int *ptr;
ptr = new int;
delete ptr;

int *ptr;
ptr = new int[10];
delete [] ptr;
```

Explicit References

- type &identifier – identifier is a variable of type *reference-to-type*
- references are lvalues
- const type &identifier – cannot change the referenced thing
- type & const identifier – not allowed
- Can pass-by-reference: cleaner syntax

Function Overloading

- In C, each function has exactly one type
- C++ allows overloading: multiple implementations for different parameter types (return type cannot be the only distinguishing type)
- Compiler chooses implementation based on types chosen

```
int sum(int a, int b) { return a+b; }
float sum(float a, float b) { return a+b; }
```

Operator Overloading

Redefine built-in operators like +, -, *, <, >, = in C++ to do what you want

```
Example:
class vector {
   Protected:
   double angle, length;
   public:
   //constructors & other member functions
   …
   vector operator +(const vector &b) {
       vector c;
       double ax = length*cos(angle);
       double bx = b.length*cos(b.angle);
       double ay = length*sin(angle);
       double by = b.length*sin(b.angle);
       double cx = ax+bx;
       double cy = ay+by;
       c.length = sqrt(cx*cx+cy*cy);
       c.angle = acos( cx/c.length );
       return c;}
};
```

```
vector c(1.5,2);
vector d(2.6,3);

//before operator overload
vector e = c.add(d);

//after operator overload
vector e = c + d;
```