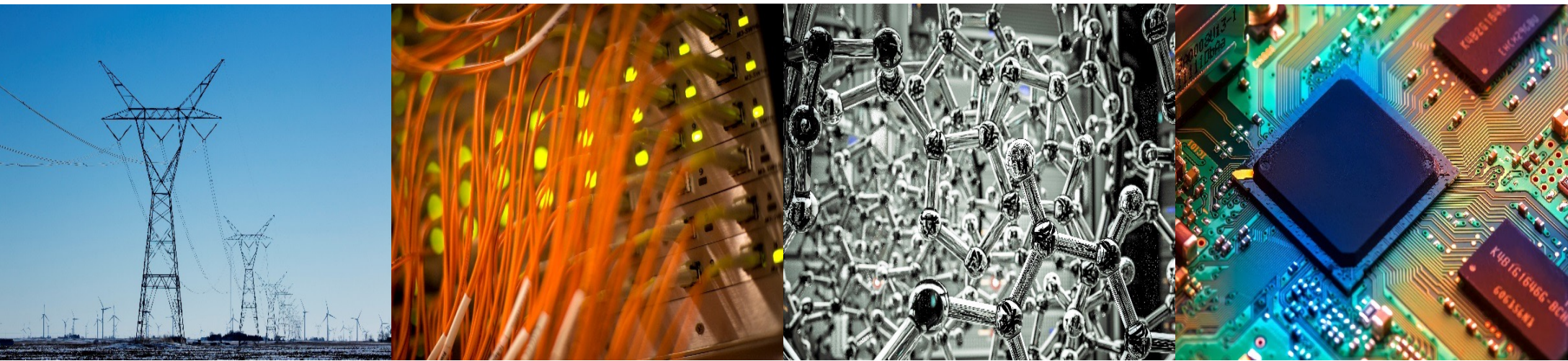


ECE 220 Computer Systems & Programming

Lecture 20 – Introduction to C++

April 9, 2026



- Quiz4 is next week
- MT2 regrade request is due next week

I ILLINOIS

Electrical & Computer Engineering

GRAINGER COLLEGE OF ENGINEERING

The type journey

Objects

struct *

struct []

struct, typedef, enum

int *, char *, float *

int[], char[], float[]

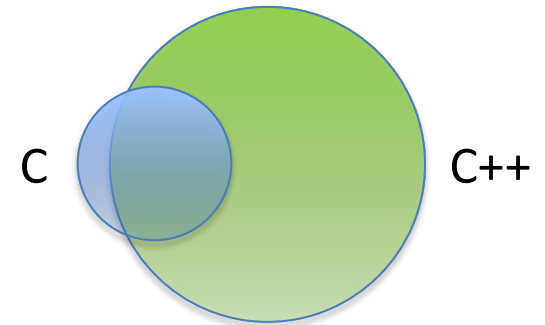
int, char, float

C++ Class & Encapsulation

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

Object-Oriented Programming (OOP) Concepts:

Encapsulation, Inheritance, Polymorphism, Abstraction



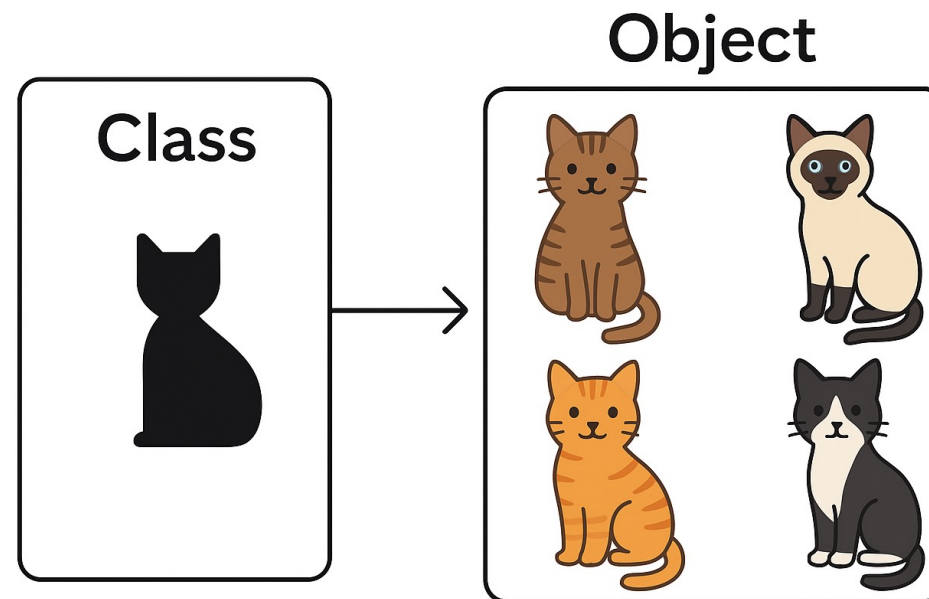
Class in C++ is similar to **Struct** in C, except it defines the data structure **AND**

- control “who” can access that data
 - provide functions specific to the class
- Can you spot the differences in C vs. C++ examples 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



Concepts Related to Class

Member functions (a.k.a. **methods**) – functions that are part of a class

Access Control

- **private** members can only be accessed by member functions (**default access**)
- **public** members can be accessed by anyone

Special Member Functions

- **constructor**: a special function that is automatically invoked to _____ a new object and _____ its data members.
- **destructor**: a special function that is automatically invoked to _____ an object (and clean up resources) when it goes outside of scope or is explicitly deleted.

Basic Input / Output

std::cin (standard input stream)

std::cout (standard output stream)

std::endl (insert new-line and flush stream)

namespace

“using namespace” directive tells compiler the subsequent code is using names in a specific namespace

Example:

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

Exercise – Writing Constructors

```
class Rectangle{
    int width_, height_;
public:
    Rectangle();
    Rectangle(int, int);
    int area() const {return width_*height_;}
};

Rectangle::Rectangle() {
//set both width_ and height_ to 0

}

Rectangle::Rectangle(int w, int h){
//set width_ to w and height_ to h

}

}
```

Exercise – Accessing Members in an Object

```
#include <iostream>
using namespace std;
int main() {
    Rectangle rect1(3,4);
    Rectangle rect2;

    //print rect1's area

    //print rect2's area

    return 0;
}
```

- What is the area of object rect1? How about rect2?
- How do we get the width/height of each object?

Dynamic Memory Allocation

new: an operator to allocate memory on the heap (similar to *malloc* in C)

delete: an operator to deallocate heap memory (similar to *free* in C)

Use **delete[]** to deallocate an array

Example:

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

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

Exercise – Accessing Objects Through Pointers

```
#include <iostream>
using namespace std;
int main() {
    Rectangle rect1(3,4);
    Rectangle *r_ptr1 = &rect1;
    //print rect1's area through r_ptr1

    Rectangle *r_ptr2 = new Rectangle(5,6);
    //print area of rectangle pointed to by r_ptr2
    if(r_ptr2 != NULL) {
    Rectangle *r_ptr3 =
        new Rectangle[2]{Rectangle(),Rectangle(2,4)};
    //print area of the 2 rectangles in the array
    if(r_ptr3 != NULL) {

    //deallocate memory

    return 0;
}
```

Function Overloading

- In C, each function has exactly one type
- C++ allows overloading – multiple implementations for **different parameter types**
- Compiler chooses implementation based on the types chosen

Example:

```
int getmin(int a, int b){  
    return (a<b)?a:b;  
}
```

```
double getmin(double a, double b){  
    return (a<b)?a:b;  
}
```

Operator Overloading

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

Example:

```
class vector {
    protective:
        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 a(1.5,2);
vector b(2.6,3);

//before operator overload
vector c = a.add(b);

//after operator overload
vector c = a + b;
```