ECE 220 Computer Systems & Programming

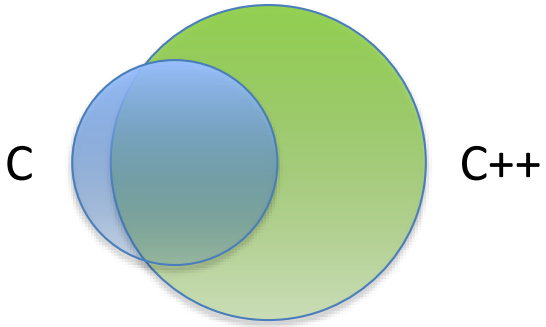
Lecture 23 – Intro to C++ and Inheritance

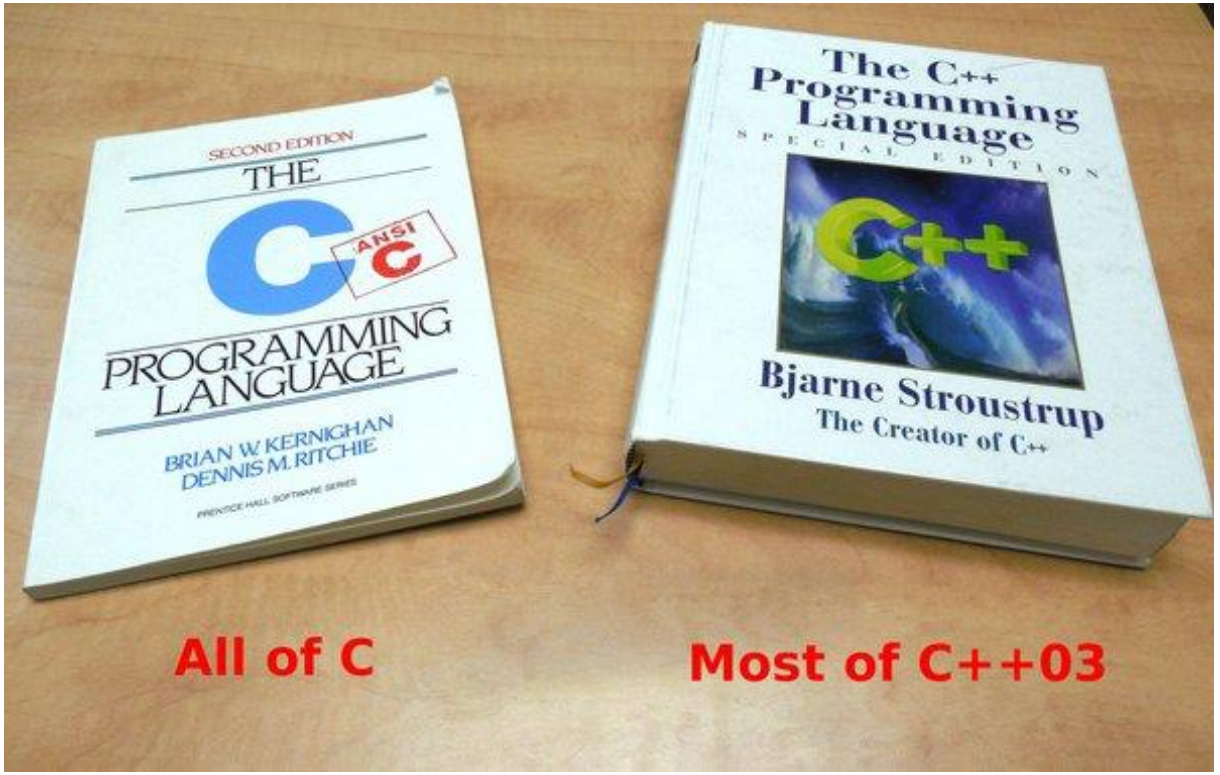
ECE ILLINOIS

ILLINOIS

C++ - Class & Encapsulation

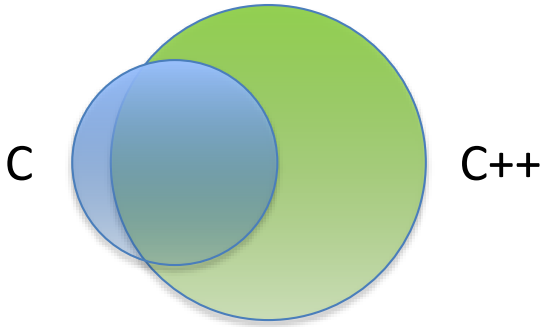
- Created in 1979 by Bjarne Stroustrup at Bell Labs, as an extension to C





C++ - Class & Encapsulation

- Created in 1979 by Bjarne Stroustrup at Bell Labs, as an extension to C

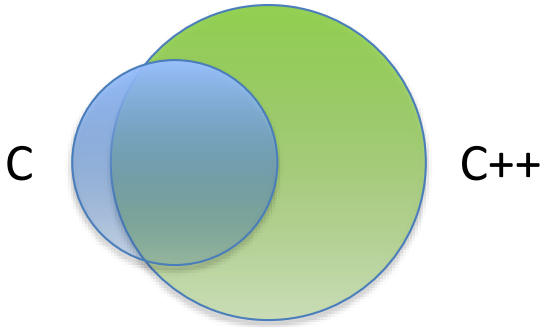


C++ - Class & Encapsulation

- 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

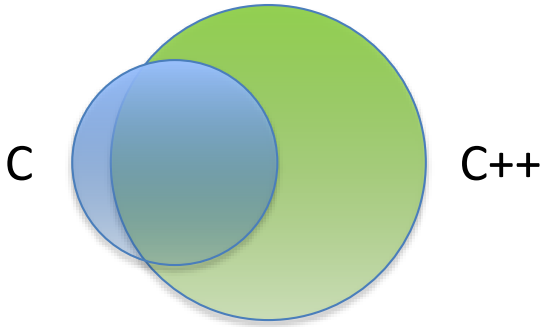


C++ - Class & Encapsulation

- 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



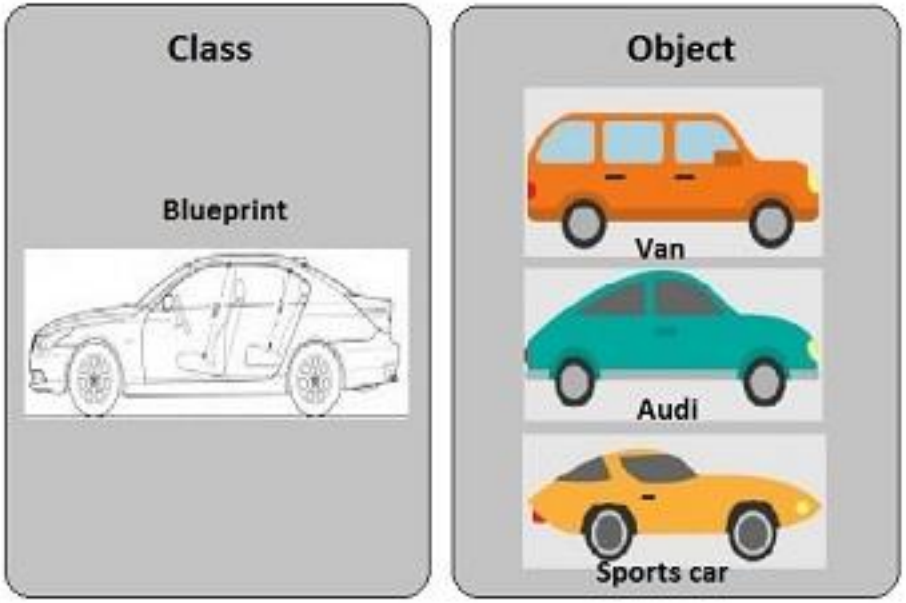
Class in C++ is similar to Struct in C, except it defines the data structure **AND**

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

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

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

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

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

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

Constructors & Destructors

- Constructor – a special member function that *creates* (initiates) a new object
- Destructor – a special member function that *deletes* an object.

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

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;
}
```

```
1  #include <iostream>
2
3  void swap(int &x, int &y)
4  {
5     int temp = x;
6     x = y;
7     y = temp;
8  }
9
10 void swap(char &x, char &y)
11 {
12    char temp = x;
13    x = y;
14    y = temp;
15 }
16
17 void swap(double &x, double &y)
18 {
19    double temp = x;
20    x = y;
21    y = temp;
22 }
23
24 int main (void)
25 {
26    // Variable declarations
27    int a = 4, b =5;
28    char c = 'c', d = 'd';
29    double x = 3.14, y = 1.41;
30
31    // Before the swaps
32    std::cout << "a = " << a << "   b = " << b << std::endl;
33    std::cout << "c = " << c << "   d = " << d << std::endl;
34    std::cout << "x = " << x << "   y = " << y << std::endl;
35
36    swap(a, b);
37    swap(c, d);
38    swap(x, y);
29
40    // After the swaps
41    std::cout << "a = " << a << "   b = " << b << std::endl;
42    std::cout << "c = " << c << "   d = " << d << std::endl;
43    std::cout << "x = " << x << "   y = " << y << std::endl;
44 }
```

1. Namespaces
2. Input Output (<<, >>, combined with cout and cin respectively)
3. Pass by reference
4. Function overloading

Dynamic Memory Allocation

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

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

Example:

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

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

Classes

```
// C++ code
class Triangle {
    double sideA;
    double sideB;
    double sideC;
};

Triangle t1;
```

```
// C code
struct Triangle {
    double sideA;
    double sideB;
    double sideC;
};

struct Triangle t2;
```

```
 1  class Triangle {
 2     double sideA;
 3     double sideB;
 4     double sideC;
 5   public:
 6     double area();
 7     double perimeter();
 8  };
 9
10  double Triangle::perimeter()
11  {
12     return sideA + sideB + sideC;
13  }
14
15  double Triangle::area()
16  {
17     double s = perimeter() / 2;
18
19     // Heron's Formula
20     return sqrt(s*(s - sideA)*(s - sideB)*(s - sideC));
21  }
```

A Method call in C++

```
1  int main(void)
2  {
3      Triangle t1;
4      double p1;
5      :
6      :
7      p1 =  t1.perimeter();
8      :
9  }
```

Access Specifiers for private member:

SideA, SideB and SideC

```
1  int main(void)
2  {
3     Triangle t1;
4     double a1;
5     :
6     :
7     t1.sideA = 10;     // This will generate an error
8     a1 =  t1.area();
9     :
10 }
```

Constructor

```
1  class Triangle {
2     double sideA;
3     double sideB;
4     double sideC;
5    public:
6     Triangle(double a, double b, double c);
7     double area();
8     double perimeter();
9  };
10
11 Triangle::Triangle(double a, double b, double c)
12 {
13    sideA = a;
14    sideB = b;
15    sideC = c;
16 }
```

Constructor

```
1  class Triangle {
2     double sideA;
3     double sideB;
4     double sideC;
5    public:
6     Triangle(double a, double b, double c);
7     double area();
8     double perimeter();
9  };
10
11 Triangle::Triangle(double a, double b, double c)
12 {
13    sideA = a;
14    sideB = b;
15    sideC = c;
16 }
```

```
1   Triangle t1(2.0, 2.0, 3.0);
2   Triangle *t2;
3
4   t2 = new Triangle(4.2, 7.8, 10.2);
```

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

- Code is Posted on Github:
- L23main.c
- L23_main.cpp

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 1

}
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?

Exercise – Pointer to a Class

```
int main(){
        Rectangle rect1(3,4);
        Rectangle *r_ptr1 = &rect1;
        //print rect1's area through r_ptr1

        Rectangle *r_ptr2, *r_ptr3;
        r_ptr2 = new Rectangle(5,6);
        //print area of rectangle pointed to by r_ptr2

        r_ptr3 = new Rectangle[2]{Rectangle(),Rectangle(2,4)};
        //print area of the 2 rectangles in the array



        //deallocate memory


        return 0;
}
```

Inheritance & Abstraction

C++ allows us to define a class based on an existing class, and the new class will inherit members of the existing class.

- the **existing** class –
- the **new** class –

A derived class inherits all base class member functions with the following exceptions:

- Constructors, destructors and copy constructors of the base class.
- Overloaded operators of the base class.
- The friend functions of the base class.