ClassesAndObjects

January 18, 2026

1 Classes and Objects

1.1 External Resources

- Classes I - https://cplusplus.com/doc/tutorial/classes/
- Classes II - https://cplusplus.com/doc/tutorial/templates/
- YouTube Video - https://youtu.be/-44fFS7atfI
- YouTube Podcast - https://youtu.be/meW8VjjpUU0
- NotebookLM learning materials - https://notebooklm.google.com/notebook/1253861c-4628-4490-a2d1-99783cadcc11

1.2 Object Oriented Programming (OOP) Concepts

- OOP (Object Oriented Programming) is a programming paradigm that uses "objects" to design software.
- A class is a blueprint for creating objects.
- An object is an instance of a class.
- Classes consists of attributes (data members) and methods (functions).
- Encapsulation is the bundling of data and methods within a class.
- **Inheritance** allows one class to inherit properties from another.
- **Polymorphism** enables objects to take multiple forms.
- **Abstraction** is the concept of hiding complex implementation details and showing only the necessary parts.
- Classes must be defined before they are used to create objects.

1.3 Defining a Class

- syntax to define a class in C++:

"cpp class ClassName { int attribute1; float attribute2; std::string attribute3; // Methods void method1() { // Method implementation } }; - by default, all members of a class are private. - usepublic:access specifier to make members accessible from outside the class. - useprivate:access specifier to restrict access to members within the class only. - useprotected:access specifier to allow access to members within the class and its derived classes - constructors are special methods that are called when an object of the class is created. - destructors are special methods that are called when an object of the class is destroyed. - usethispointer to refer to the current object within class methods. - usegettersandsetters' to access and modify private attributes of a class.

[10]:
```
#include <iostream>
#include <string>
using namespace std;
```

[2]:
```
// Define a Rectangle class with methods to calculate area and perimeter
class Rectangle {
  private:
    double length;
    double width;

  public:
    // Constructor
    Rectangle(double l, double w){
      this->length = l;
      this->width = w;
    }

    // Method to calculate area
    double area() {
      return this->length * this->width;
    }

    // Method to calculate perimeter
    double perimeter() {
      return 2 * (this->length + this->width);
    }

    // getter for length
    double get_length() {
      return this->length;
    }
    // getter for width
    double get_width() {
      return this->width;
    }

    // setter for length
    void set_length(double l) {
      this->length = l;
    }
    // setter for width
    void set_width(double w) {
      this->width = w;
    }
    // Destructor
    ~Rectangle() {}
};
```

[3]:
```cpp
// instantiate and use the Rectangle class
Rectangle rect(5.0, 3.0);
cout << "Area: " << rect.area() << endl;
cout << "Perimeter: " << rect.perimeter() << endl;
```

```
Area: 15
Perimeter: 16
```

[4]:
```cpp
// Using pointers to create Rectangle objects
Rectangle * rectPtr = new Rectangle(4.0, 2.0);
cout << "Area (using pointer): " << rectPtr->area() << endl;
cout << "Perimeter (using pointer): " << rectPtr->perimeter() << endl;
delete rectPtr;
```

```
Area (using pointer): 8
Perimeter (using pointer): 12
```

1.4 Overloading Operators

- operator overloading allows you to define custom behavior for operators when they are used with objects of a class.
- syntax to overload an operator in a class:
  - `friend` function or member function
  - `operator operator_symbol` followed by parameters and return type.

```cpp
class ClassName {
public:
  // Overload + operator
  ClassName operator+(const ClassName &other) {
    ClassName result;
    // Custom addition logic
    return result;
  }
  // Overload == operator
  bool operator==(const ClassName &other) const {
    return (this->x == other.x) && (this->y == other.y);
  }
  // Overload << operator
  friend std::ostream& operator<<(std::ostream& os, const ClassName& obj) {
    os << "Custom output format";
    return os;
  }
};
```

- use operator overloading to enhance code readability and maintainability.
- `friend` functions can access private and protected members of the class.
- `friend` functions are not member functions of the class.
- see `demos/classes/points/` for examples of operator overloading.
  - doctest tests for classes and operator overloading.
- separate class definitions and implementations into header (`.h`) and source (`.cpp`) files for better organization.

[ ]: