Object-Oriented Programming

- In C++ classes provide the functionality necessary to use *object-oriented programming*
  - OOP is a particular way of organizing computer programs
  - It doesn't allow you to do anything you couldn't already do, but it makes it arguably more efficient
  - OOP is by far the dominant software engineering practice in the last two decades

- Classes combine data and functionality
  - Class members can store structured data, as we've seen
  - Class members can also be functions
    - Class-specific functions are called ***constructor methods***

The `string` class

1. Character array

    char char_array[100];

‘a’	‘e’	‘i’	‘o’	‘u’	‘y’	‘\0’			

0	1	2	3	4	5	6	7	8	9

    char_array[0] = ‘a’; ……
    char_array[6] = ‘\0’;

2. String type

Note: include the string library.

    string str_var;
    str_var = “aeiouy”;

The `string` class

- The string class has private data members to store the characters that make up a string
  - It probably uses an array, although it doesn't have to
  - It probably has ints to keep track of the size of the array and the number of characters
- The string class has public ***(constructor) methods*** to do stuff
  - Return the number of characters in the internal storage

    ```
    int len();
    ```
  - Append the characters in s to the internal storage

    ```
    void append( string s );
    ```
  - returns the position of s within the internal storage

    ```
    int find( string s);
    ```

Date class

- What data should the Date class store?

- Date: 2020-11-11
- Components: int year, int month, int day

```
// constant variable: PI; MAX_LENGHT = 10;
Class Date{
        public:
                int year;
                int month;
                int day;
                Date();
                Date(int yr, int mth, int d);
                void print();
                void before(Date d1, Date d2);
};
```

```
// define function
void print_func(){
    // do whatever we want there;
}

// same as for constructor methods
```

Date class

```
void Date::print(){
        cout << year << “/” << month <<“/” << day << endl;
}

// we want to know if d2 is before d1 or not.
void Date::before(Date d1, Date d2){
        //if (d1.year > d2.year) {cout << “true” << endl;}
        //else if (d1.year == d2.year & d1.month > d2.month) {cout << “true” << endl;}
        //else if (d1.year == d2.year & d1.month == d2.month &
        //            d1.day > d2.day) {cout << “true” << endl;}
        //else{cout << “false” << endl;}

        if (d1.year > d2.year || (d1.year == d2.year & d1.month > d2.month) ||
                    (d1.year == d2.year & d1.month == d2.month &
                    d1.day > d2.day)) {cout << “true” << endl;}
        else {cout << “false” << endl;}
}
```

Date class

- What functionality (***constructor methods***) would we like Dates to have?

```
// one way to initialize the variables in a class.
Date:: Date(){
        year = 2020;
        month = 10;
        day = 15;
}

Date::Date(int yr, int mth, int d){
        year = yr;
        month = mth;
        day = d;
}
```

Date class

```
// in main.cpp
Date date_obj1;  // denote date_obj1 as an object and initialized.
Date date_obj2(2020, 11, 11);  // denote date_obj2 as an object and
initialized.

cout << date_obj1.month << endl;  // 10
cout << date_obj2.month << endl;  // 11
```