MERRIMACK COLLEGE

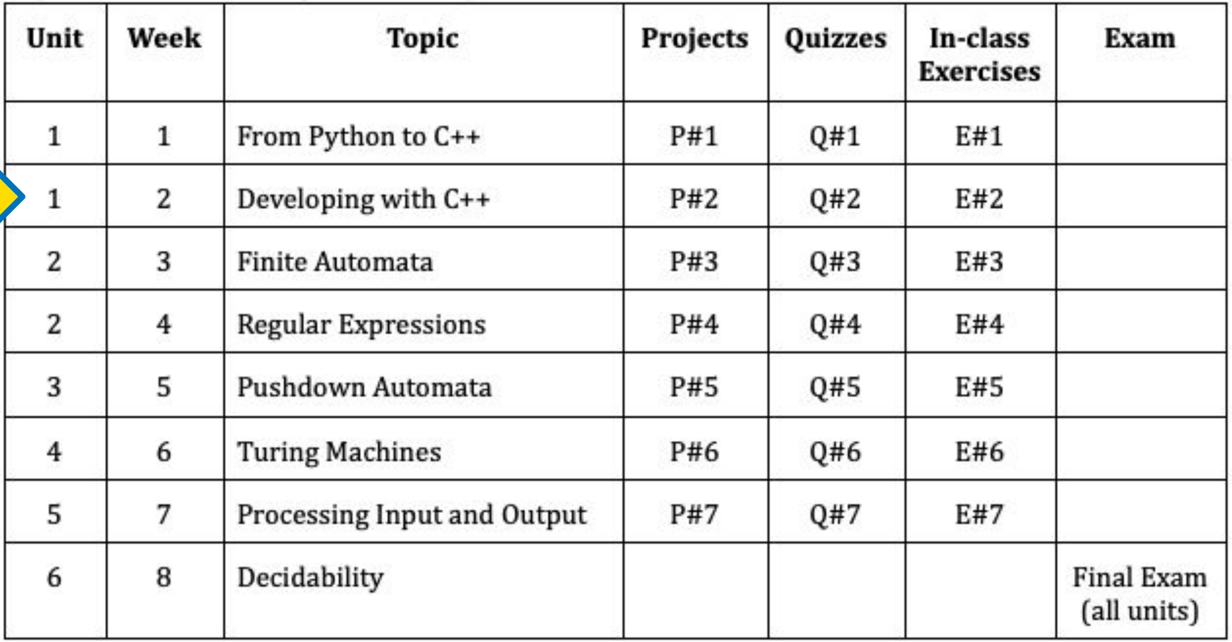
# CSC 6033 Week 2

Developing with C++

Languages, Automata, and Decidability - Dr. Paulo Fernandes

# Timeline

Where are we?



| Unit | Week | Topic | Projects | Quizzes | In-class Exercises | Exam |
|---|---|---|---|---|---|---|
| 1 | 1 | From Python to C++ | P#1 | Q#1 | E#1 | |
| 1 | 2 | Developing with C++ | P#2 | Q#2 | E#2 | |
| 2 | 3 | Finite Automata | P#3 | Q#3 | E#3 | |
| 2 | 4 | Regular Expressions | P#4 | Q#4 | E#4 | |
| 3 | 5 | Pushdown Automata | P#5 | Q#5 | E#5 | |
| 4 | 6 | Turing Machines | P#6 | Q#6 | E#6 | |
| 5 | 7 | Processing Input and Output | P#7 | Q#7 | E#7 | |
| 6 | 8 | Decidability | | | | Final Exam (all units) |

MERRIMACK COLLEGE

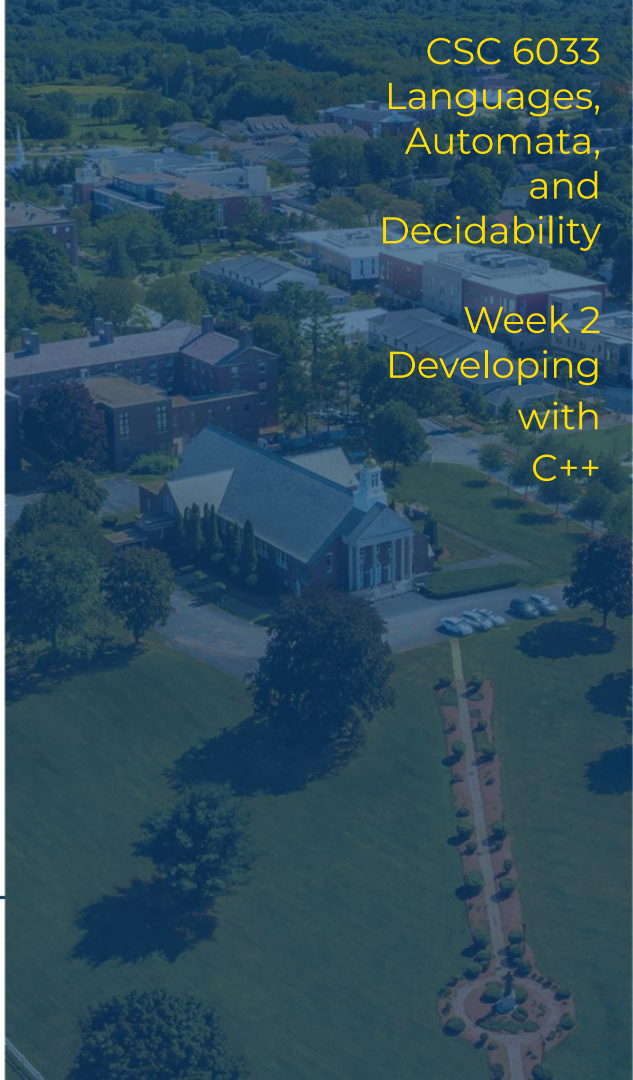
Don't let tasks pile up!

# Presentation Agenda

Week 2

- Classes and Objects
  1. From ***struct*** to ***class***
  2. Methods
  3. Encapsulation
  4. Inheritance
  5. Polymorphism
- This Week's tasks

MERRIMACK COLLEGE



CSC 6033 Languages, Automata, and Decidability

Week 2 Developing with C++

# Classes and Objects

C++, once called C with Classes

Object-oriented concepts were the reason to create C++ language.

In C++, unlike Java, there are imperative portions and there is data structures encapsulated as objects.

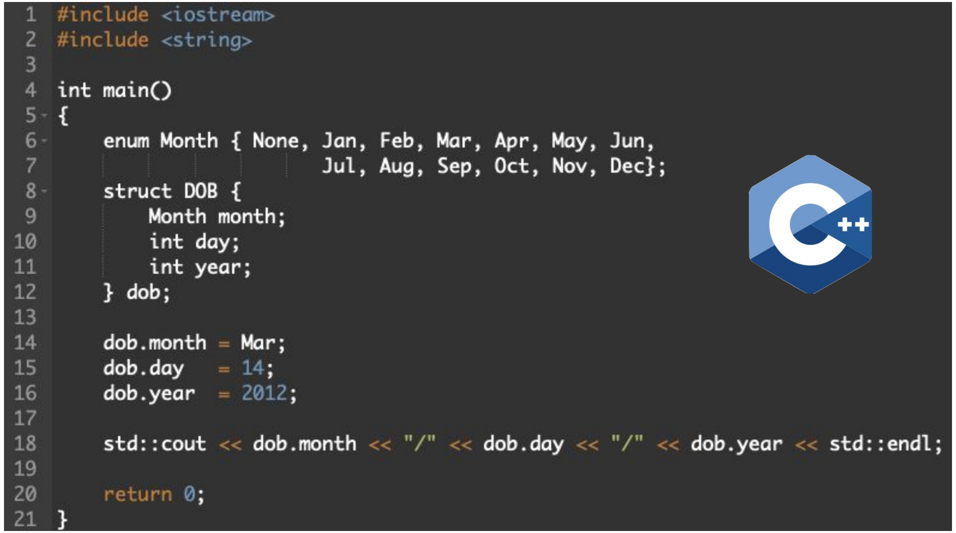
- **From *struct* to *class***
- Methods
- Encapsulation
- Inheritance
- Polymorphism

# From struct to class

The command struct, already available in C, is a binder for variables. While in an array all elements are of the same type, in a structure, each element can be of a different type.

For example:

- We define an enumeration (enum) with the three letter code for months (***Month***);
- We define a variable ***dob*** that is binding a month, a day, and an year.

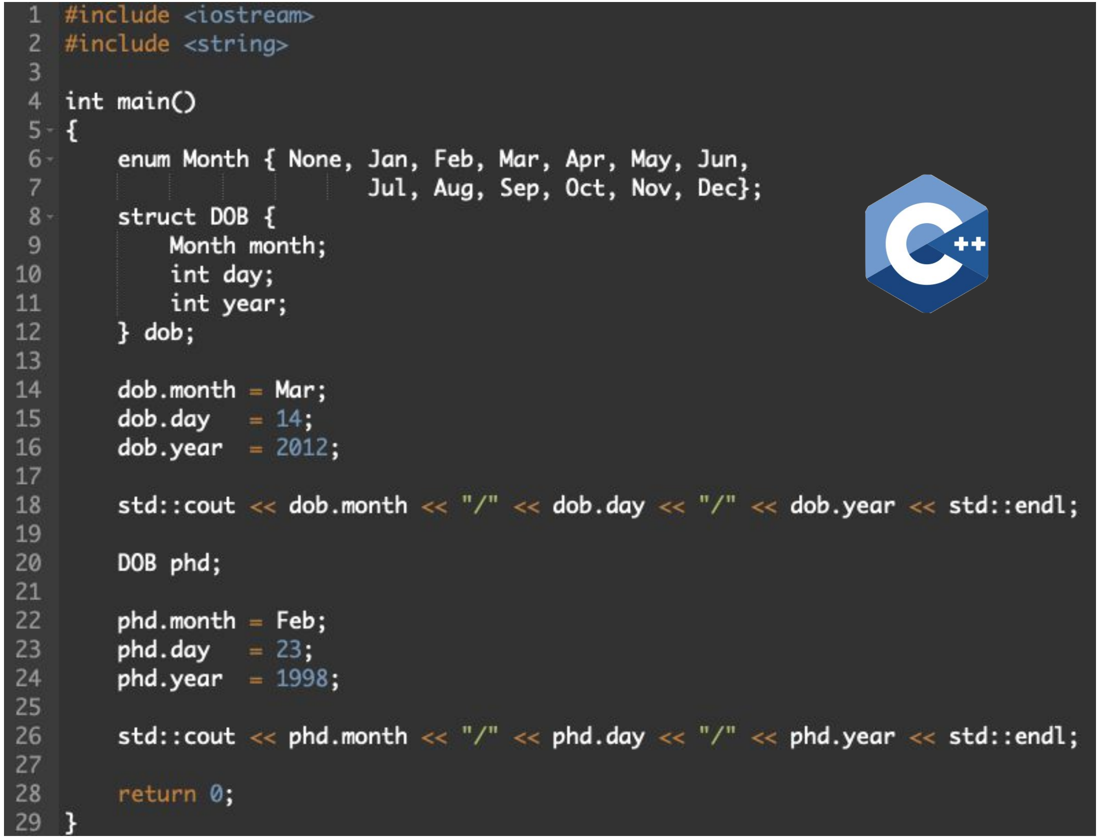


```cpp
#include <iostream>
#include <string>

int main()
{
    enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
                       Jul, Aug, Sep, Oct, Nov, Dec};
    struct DOB {
        Month month;
        int day;
        int year;
    } dob;

    dob.month = Mar;
    dob.day   = 14;
    dob.year  = 2012;

    std::cout << dob.month << "/" << dob.day << "/" << dob.year << std::endl;

    return 0;
}
```

Text: C++ Enumerations (enum).

# From struct to class

Pushing the example further we define a new data type DOB that can be used any time later…



```cpp
#include <iostream>
#include <string>

int main()
{
    enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
                       Jul, Aug, Sep, Oct, Nov, Dec};
    struct DOB {
        Month month;
        int day;
        int year;
    } dob;

    dob.month = Mar;
    dob.day   = 14;
    dob.year  = 2012;

    std::cout << dob.month << "/" << dob.day << "/" << dob.year << std::endl;

    DOB phd;

    phd.month = Feb;
    phd.day   = 23;
    phd.year  = 1998;

    std::cout << phd.month << "/" << phd.day << "/" << phd.year << std::endl;

    return 0;
}
```

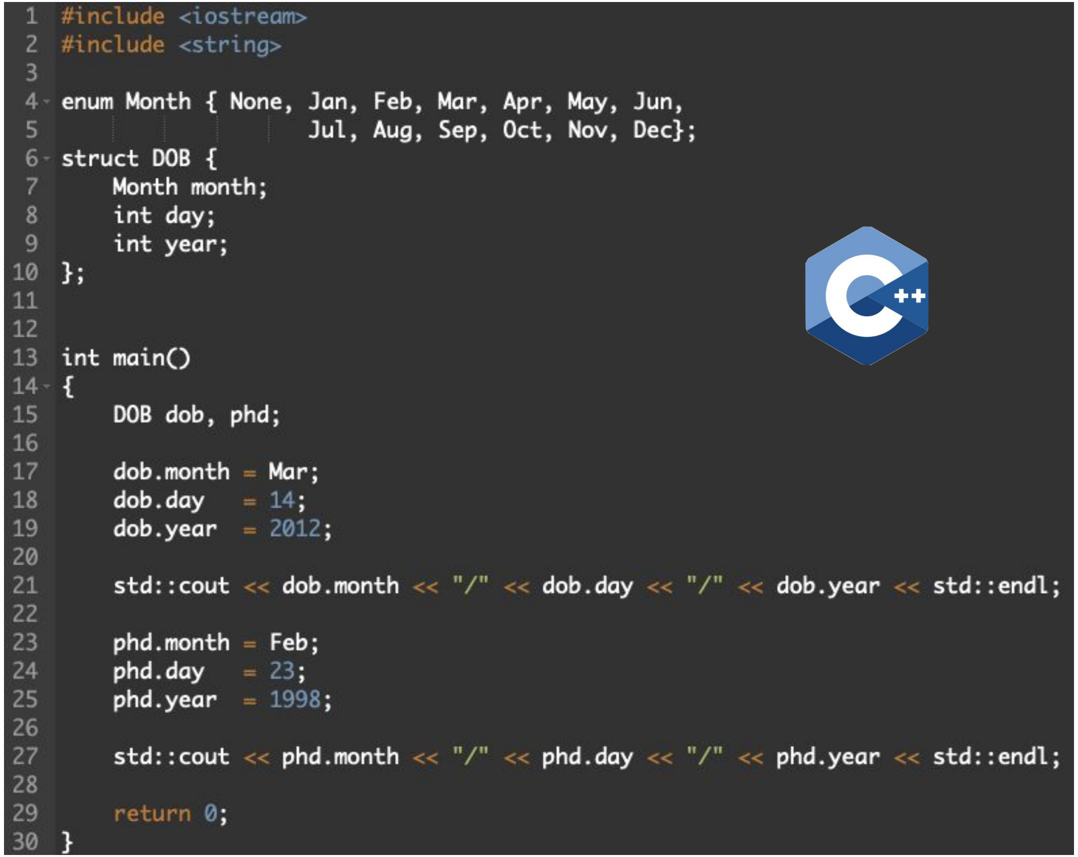
Text: C++ Structures (struct).

# From struct to class

… or even defining it to be used generally elsewhere.

Remember:

- definitions can be used within a namespace.



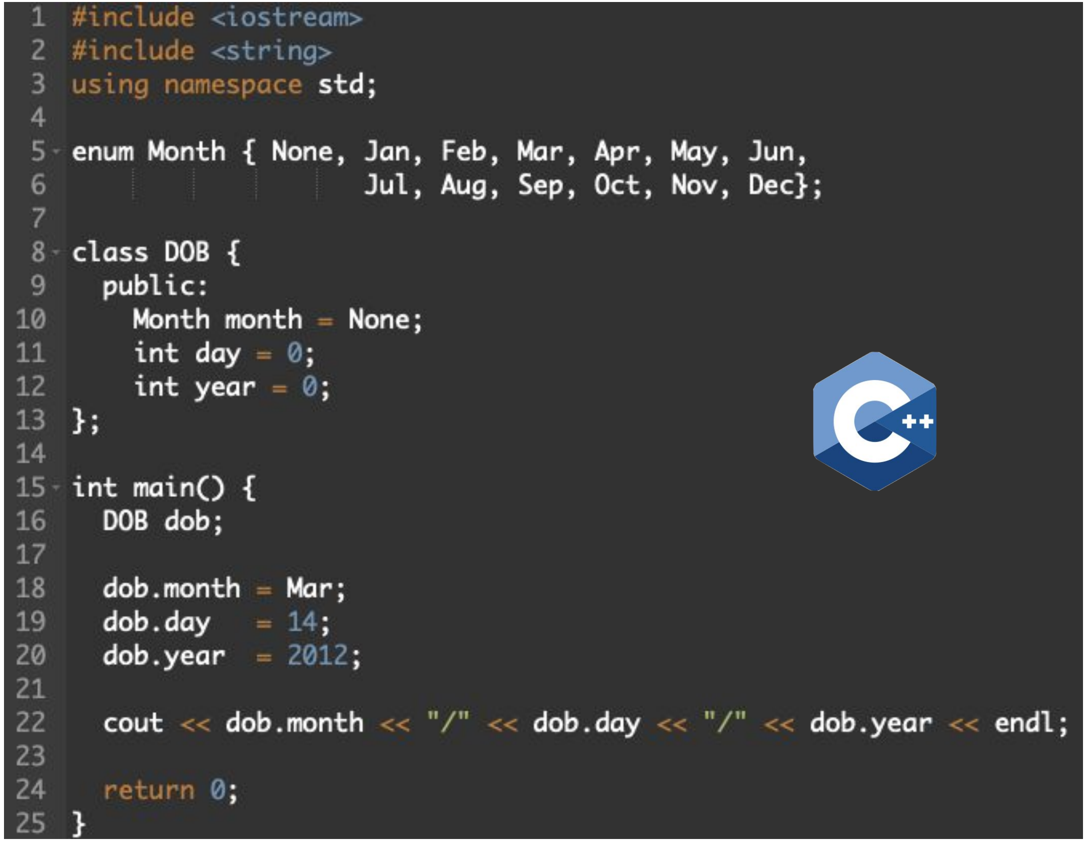
```cpp
#include <iostream>
#include <string>

enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
                   Jul, Aug, Sep, Oct, Nov, Dec};
struct DOB {
    Month month;
    int day;
    int year;
};


int main()
{
    DOB dob, phd;

    dob.month = Mar;
    dob.day   = 14;
    dob.year  = 2012;

    std::cout << dob.month << "/" << dob.day << "/" << dob.year << std::endl;

    phd.month = Feb;
    phd.day   = 23;
    phd.year  = 1998;

    std::cout << phd.month << "/" << phd.day << "/" << phd.year << std::endl;

    return 0;
}
```

A structure is an elegant way to group, but it is not an object, it has no methods!

# From struct to class

Classes are the regular way to define objects.

The class ***DOB*** becomes a data type and while declaring a variable ***dob*** in line 16, it becomes an object of class ***DOB***.



```
 1 #include <iostream>
 2 #include <string>
 3 using namespace std;
 4
 5 enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
 6                    Jul, Aug, Sep, Oct, Nov, Dec};
 7
 8 class DOB {
 9   public:
10     Month month = None;
11     int day = 0;
12     int year = 0;
13 };
14
15 int main() {
16   DOB dob;
17
18   dob.month = Mar;
19   dob.day   = 14;
20   dob.year  = 2012;
21
22   cout << dob.month << "/" << dob.day << "/" << dob.year << endl;
23
24   return 0;
25 }
```

A ***class*** as this (no methods) is just like a ***struct***.

# Classes and Objects

C++, once called C with Classes

Object-oriented concepts were the reason to create C++ language.

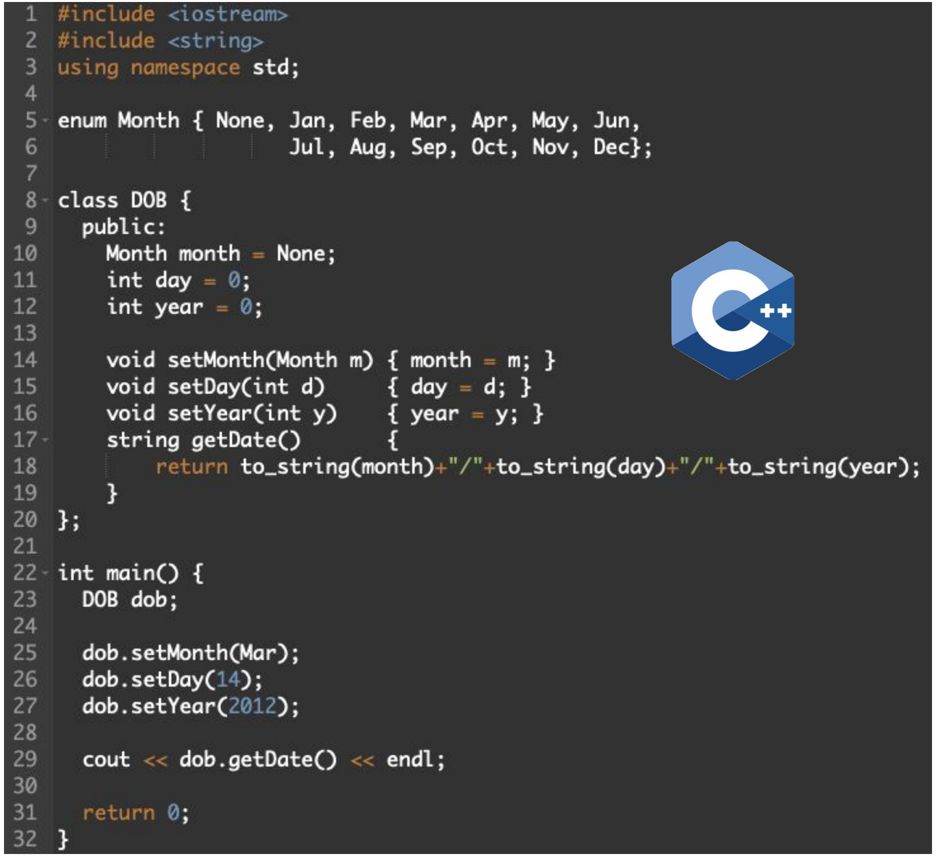
In C++, unlike Java, there are imperative portions and there is data structures encapsulated as objects.

- From *struct* to *class*
- **Methods**
- Encapsulation
- Inheritance
- Polymorphism

# Methods

Classes are the regular way to define objects.

Methods can affect or access the instance variables of an object.



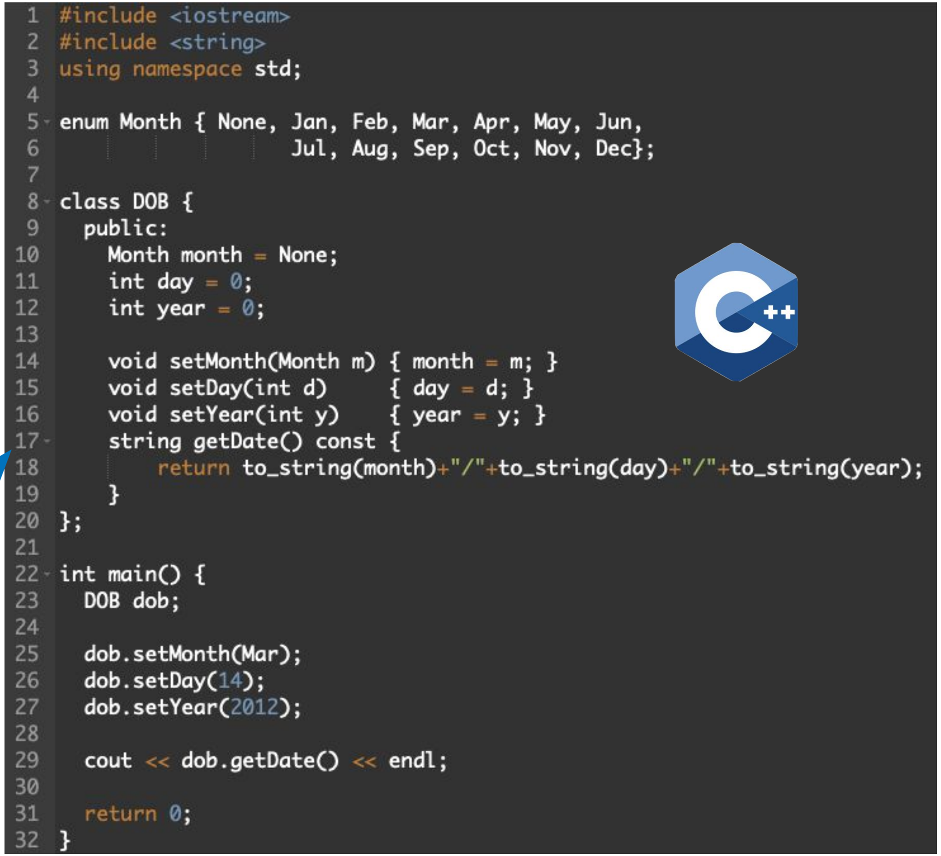
```
#include <iostream>
#include <string>
using namespace std;

enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
                   Jul, Aug, Sep, Oct, Nov, Dec};

class DOB {
  public:
    Month month = None;
    int day = 0;
    int year = 0;

    void setMonth(Month m) { month = m; }
    void setDay(int d)     { day = d; }
    void setYear(int y)    { year = y; }
    string getDate()       {
        return to_string(month)+"/"+to_string(day)+"/"+to_string(year);
    }
};

int main() {
  DOB dob;

  dob.setMonth(Mar);
  dob.setDay(14);
  dob.setYear(2012);

  cout << dob.getDate() << endl;

  return 0;
}
```

Text: C++ Classes and Objects.

# Methods

If a method just access the contents of the object it is an accessor and receives the mention ***const*** after the declaration.

This is the case for the method ***getDate()*** at line 17.



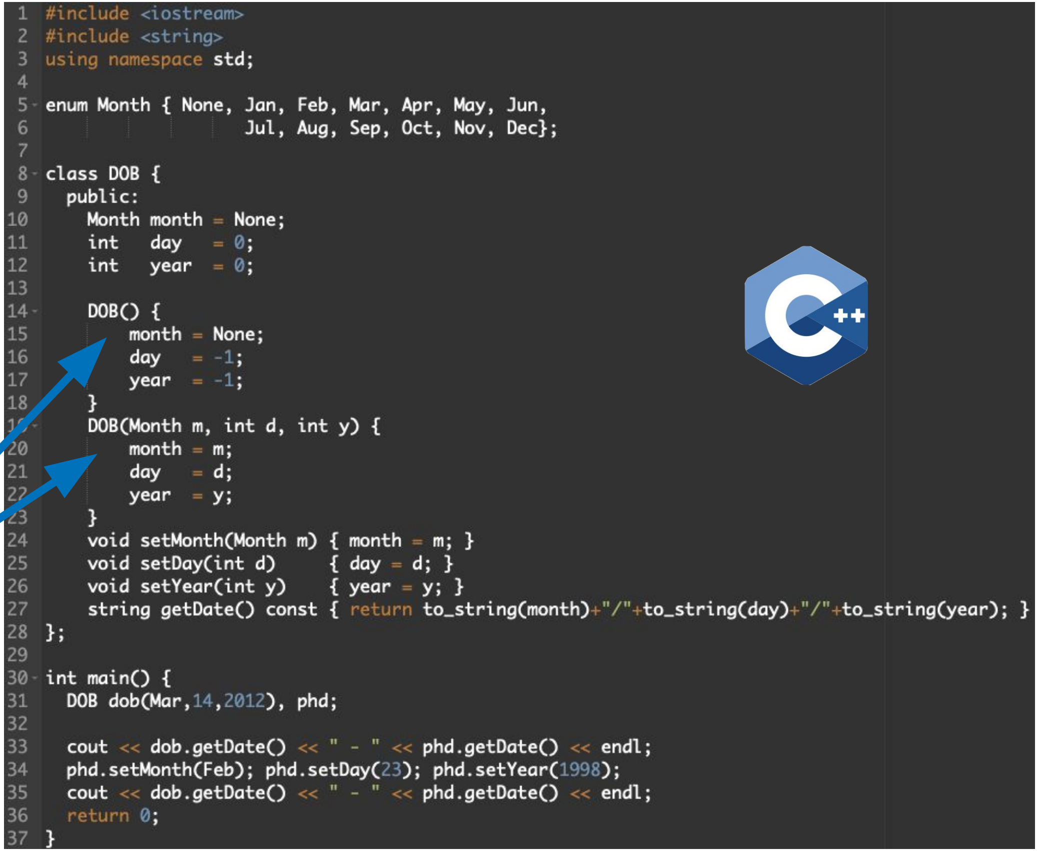
```
 1  #include <iostream>
 2  #include <string>
 3  using namespace std;
 4
 5  enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
 6                     Jul, Aug, Sep, Oct, Nov, Dec};
 7
 8  class DOB {
 9    public:
10      Month month = None;
11      int day = 0;
12      int year = 0;
13
14      void setMonth(Month m) { month = m; }
15      void setDay(int d)     { day = d; }
16      void setYear(int y)    { year = y; }
17      string getDate() const {
18          return to_string(month)+"/"+to_string(day)+"/"+to_string(year);
19      }
20  };
21
22  int main() {
23    DOB dob;
24
25    dob.setMonth(Mar);
26    dob.setDay(14);
27    dob.setYear(2012);
28
29    cout << dob.getDate() << endl;
30
31    return 0;
32  }
```

Text: Const member functions in C++.

# Methods

A special kind of method are the constructors. In C++ it is possible to define several versions of constructors if they have different sets of input parameters (*overloading*).

In this example, there is an empty constructor (line 14) and a parametrized one (line 19).



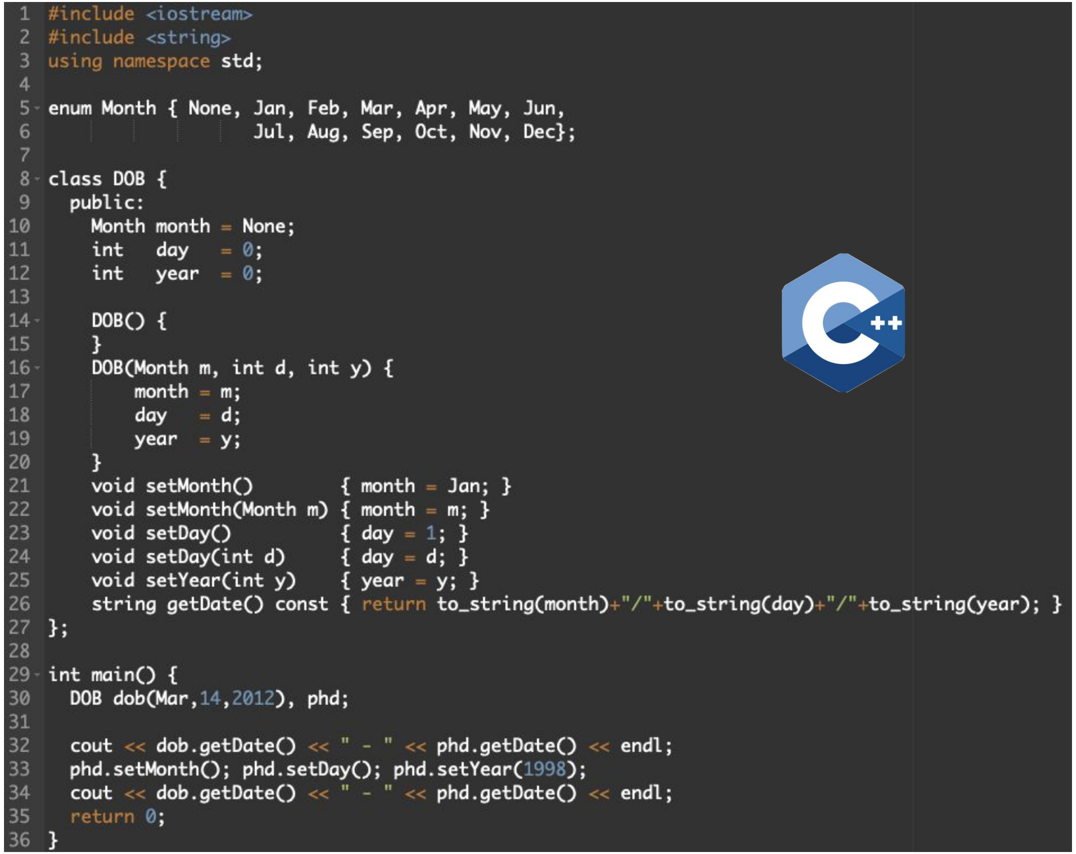
```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
6                     Jul, Aug, Sep, Oct, Nov, Dec};
7
8  class DOB {
9    public:
10     Month month = None;
11     int   day   = 0;
12     int   year  = 0;
13
14     DOB() {
15         month = None;
16         day   = -1;
17         year  = -1;
18     }
19     DOB(Month m, int d, int y) {
20         month = m;
21         day   = d;
22         year  = y;
23     }
24     void setMonth(Month m) { month = m; }
25     void setDay(int d)     { day = d; }
26     void setYear(int y)    { year = y; }
27     string getDate() const { return to_string(month)+"/"+to_string(day)+"/"+to_string(year); }
28 };
29
30 int main() {
31   DOB dob(Mar,14,2012), phd;
32
33   cout << dob.getDate() << " - " << phd.getDate() << endl;
34   phd.setMonth(Feb); phd.setDay(23); phd.setYear(1998);
35   cout << dob.getDate() << " - " << phd.getDate() << endl;
36   return 0;
37 }
```

Text: C++ Constructors.

# Methods

*Overloading* of methods is also allowed for non-constructor methods.

In this example, there are overloads of the methods ***setMonth*** (lines 21 and 22) and ***setDay*** (lines 23 and 24) setting, respectively, the first month (***Jan***) and the first day (***1***).



```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
6                      Jul, Aug, Sep, Oct, Nov, Dec};
7
8  class DOB {
9    public:
10     Month month = None;
11     int   day   = 0;
12     int   year  = 0;
13
14     DOB() {
15     }
16     DOB(Month m, int d, int y) {
17         month = m;
18         day   = d;
19         year  = y;
20     }
21     void setMonth()        { month = Jan; }
22     void setMonth(Month m) { month = m; }
23     void setDay()          { day = 1; }
24     void setDay(int d)     { day = d; }
25     void setYear(int y)    { year = y; }
26     string getDate() const { return to_string(month)+"/"+to_string(day)+"/"+to_string(year); }
27 };
28
29 int main() {
30   DOB dob(Mar,14,2012), phd;
31
32   cout << dob.getDate() << " - " << phd.getDate() << endl;
33   phd.setMonth(); phd.setDay(); phd.setYear(1998);
34   cout << dob.getDate() << " - " << phd.getDate() << endl;
35   return 0;
36 }
```

Text: C++ Function Overloading.

# Classes and Objects

C++, once called C with Classes

Object-oriented concepts were the reason to create C++ language.

In C++, unlike Java, there are imperative portions and there is data structures encapsulated as objects.

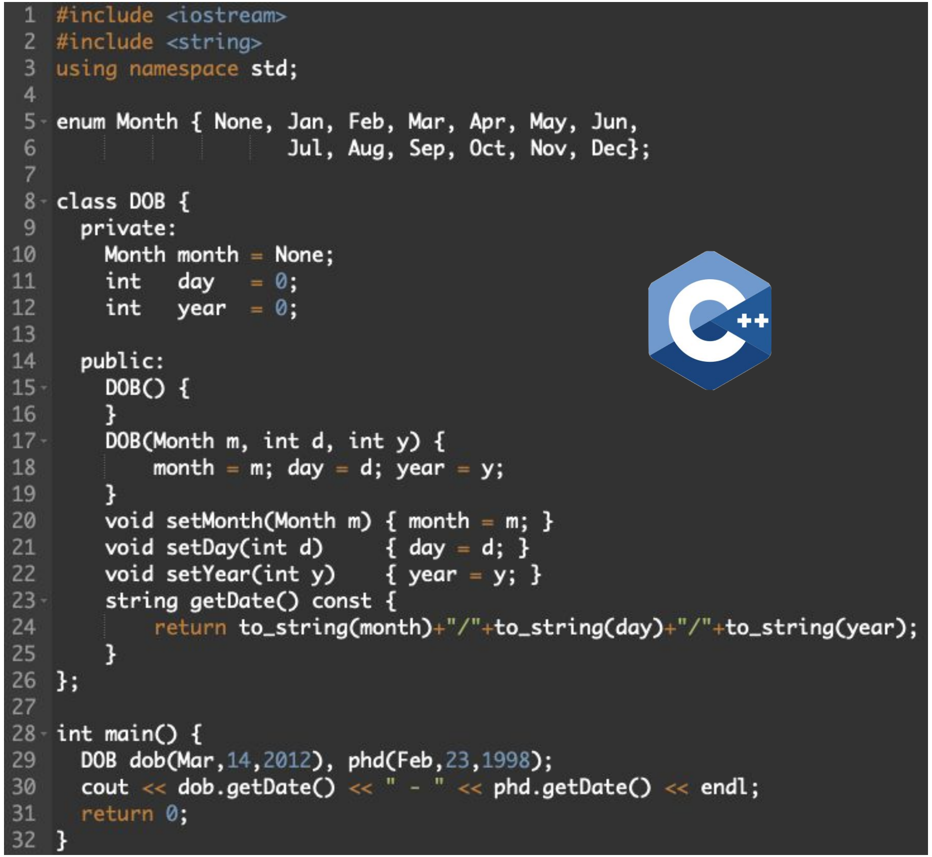
- From *struct* to *class*
- Methods
- **Encapsulation**
- Inheritance
- Polymorphism

# Encapsulation

In C++ classes you can choose what will be hidden from class users and what is not.

- It can be used only in the class, declare: ***private***;
- It can be used outside the class, declare: ***public***;

This can be applied to instance variables and methods using the group declarations (as many as wanted).



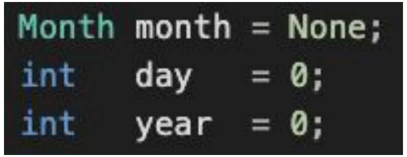
```cpp
#include <iostream>
#include <string>
using namespace std;

enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
                   Jul, Aug, Sep, Oct, Nov, Dec};

class DOB {
  private:
    Month month = None;
    int   day   = 0;
    int   year  = 0;

  public:
    DOB() {
    }
    DOB(Month m, int d, int y) {
        month = m; day = d; year = y;
    }
    void setMonth(Month m) { month = m; }
    void setDay(int d)     { day = d; }
    void setYear(int y)    { year = y; }
    string getDate() const {
        return to_string(month)+"/"+to_string(day)+"/"+to_string(year);
    }
};

int main() {
  DOB dob(Mar,14,2012), phd(Feb,23,1998);
  cout << dob.getDate() << " - " << phd.getDate() << endl;
  return 0;
}
```

MERRIMACK COLLEGE

Text: C++ Encapsulation.

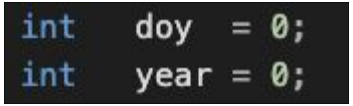
# Encapsulation

To exemplify the benefits of "hiding", we will change the internal structure for the class ***DOB***.



```
Month month = None;
int   day   = 0;
int   year  = 0;
```

Instead of having three instance variables (***month***, ***day***, and ***year***). We will consider a single Integer (going from 0 to 366) to represent the day and month together, called *Day of the Year* (*doy*).



```
int    doy  = 0;
int    year = 0;
```

For example:

- 0 to no date;
- 3 to January 3;
- 44 to February 13; and
- 63 to March 3 on leap years or March 4 otherwise.

Text: Day of the Year (DOY) calendar.

# Encapsulation

We will create two internal methods:

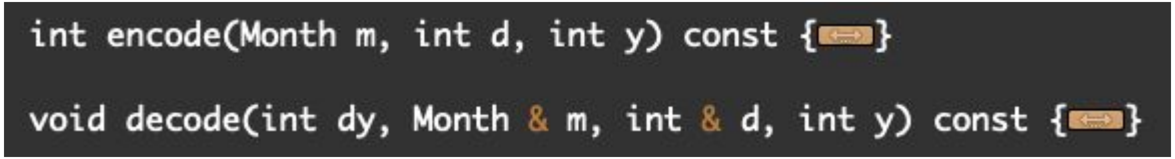
- Encode that receives the month, the day, and the year, delivering the day of the year;
- Decode that receives the day of the year and year, delivering the month and the day.

Input parameters in C++ can be passed:

- *by value* - they serve only as input parameters;
- *by reference* - they serve as both input and output parameters.

To pass a parameter by reference you actually pass its address reference using the symbol &.

For example, ***Month & m*** and ***int & d*** at decode method.

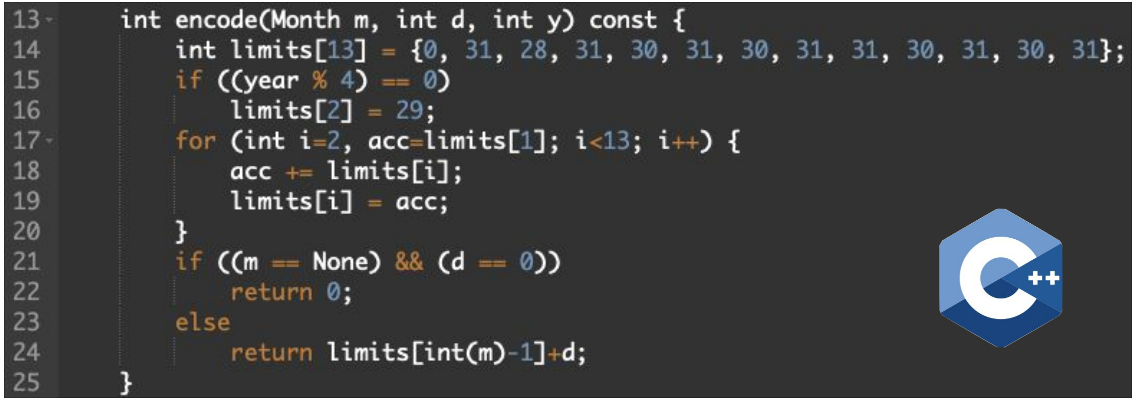


```
int encode(Month m, int d, int y) const {⟺}

void decode(int dy, Month & m, int & d, int y) const {⟺}
```

Text: C++ Functions - Pass By Reference.

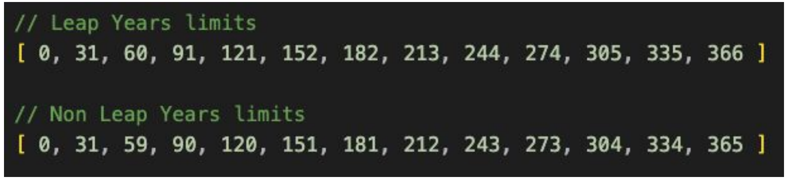
# Encapsulation

The method ***encode*** receives the month ***m***, the day ***d***, and the year ***y***, delivering the day of the year (***return***).



```
int encode(Month m, int d, int y) const {
    int limits[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if ((year % 4) == 0)
        limits[2] = 29;
    for (int i=2, acc=limits[1]; i<13; i++) {
        acc += limits[i];
        limits[i] = acc;
    }
    if ((m == None) && (d == 0))
        return 0;
    else
        return limits[int(m)-1]+d;
}
```

- Create a vector with 13 elements, 0 for the first, then the number of days of each month (if leap year Feb has 29);
- This vector becomes the *doy* of the last day of each month, this vector is called ***limits***;
- It computes doy as the end of the previous of month ***m***, plus the day ***d***.



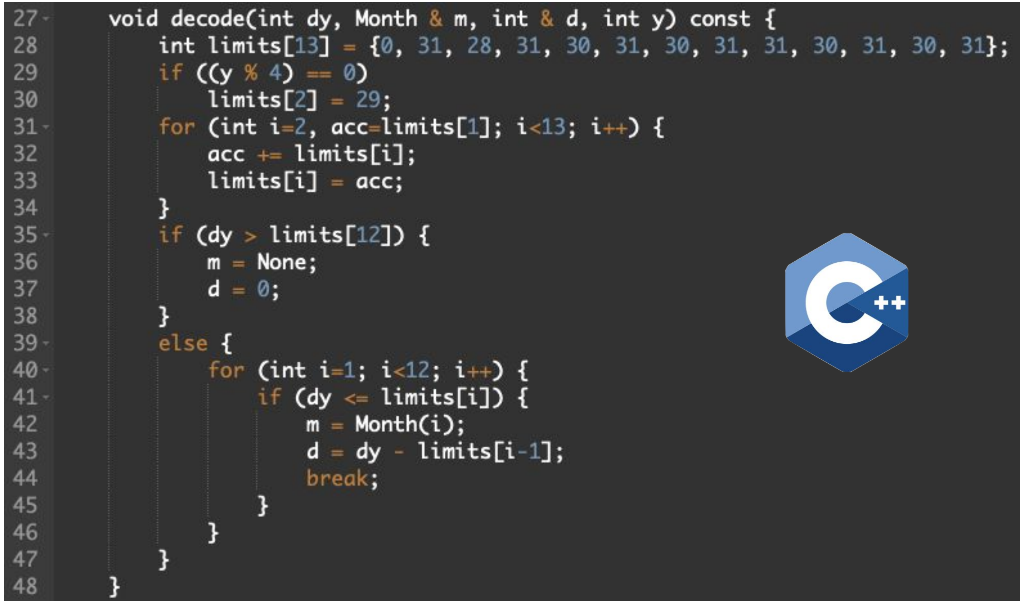
```
// Leap Years limits
[ 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 ]

// Non Leap Years limits
[ 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 ]
```

Decode: ***m***, ***d***, ***y*** => ***return*** doy

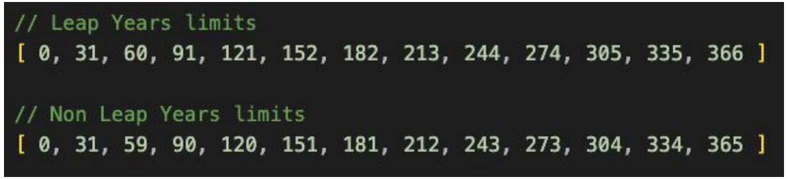
# Encapsulation

The method ***decode*** receives the the day of the year ***dy*** and the year ***y***, delivering month ***m*** and the day ***d***.

- The vector ***limits*** is created in the same way as in the ***decode*** method;
- It computes ***m*** as the previous month of the one with the end greater than ***dy***, and ***d*** as the remainder days of the chosen ***m***.



```cpp
void decode(int dy, Month & m, int & d, int y) const {
    int limits[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if ((y % 4) == 0)
        limits[2] = 29;
    for (int i=2, acc=limits[1]; i<13; i++) {
        acc += limits[i];
        limits[i] = acc;
    }
    if (dy > limits[12]) {
        m = None;
        d = 0;
    }
    else {
        for (int i=1; i<12; i++) {
            if (dy <= limits[i]) {
                m = Month(i);
                d = dy - limits[i-1];
                break;
            }
        }
    }
}
```



```
// Leap Years limits
[ 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335, 366 ]

// Non Leap Years limits
[ 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 ]
```

Decode: ***dy***, ***y*** => ***m***, ***d***

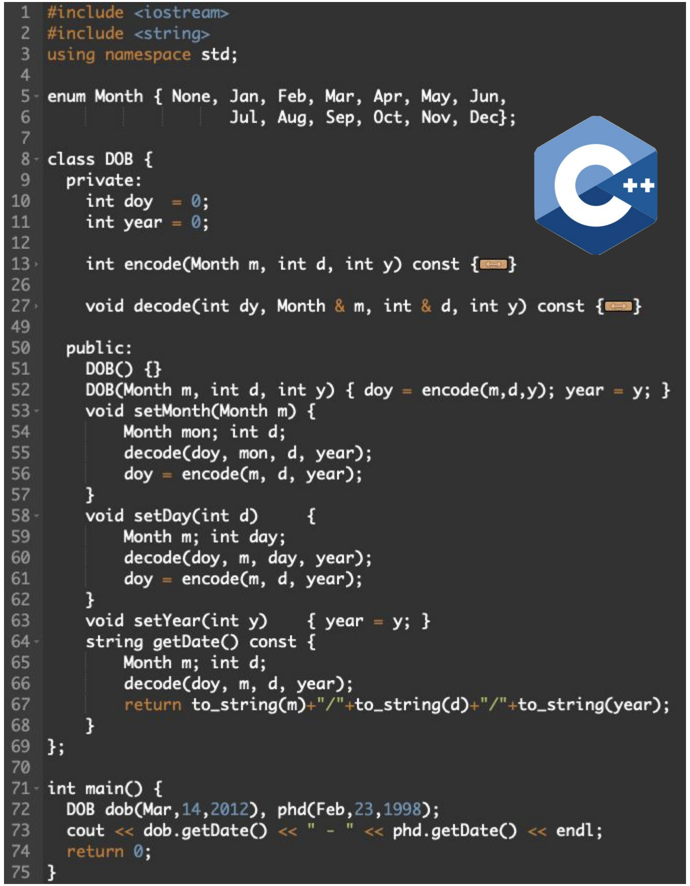
# Encapsulation

The class ***DOB*** has instance variables ***doy*** and ***year***, which are *private*. The methods ***encode*** and ***decode*** are also *private*.

The parameterized constructor still receives month ***m***, day ***d***, and year ***y***, but saves into instance variables as date of the year ***doy*** and ***year***.

The methods ***setMonth***, ***setDay***, ***getDate*** change the implementation, but not the call. The methods ***setYear*** does not change.

The usage of the objects remains identical.



```cpp
#include <iostream>
#include <string>
using namespace std;

enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
                   Jul, Aug, Sep, Oct, Nov, Dec};

class DOB {
  private:
    int doy  = 0;
    int year = 0;

    int encode(Month m, int d, int y) const {↔}

    void decode(int dy, Month & m, int & d, int y) const {↔}

  public:
    DOB() {}
    DOB(Month m, int d, int y) { doy = encode(m,d,y); year = y; }
    void setMonth(Month m) {
        Month mon; int d;
        decode(doy, mon, d, year);
        doy = encode(m, d, year);
    }
    void setDay(int d)     {
        Month m; int day;
        decode(doy, m, day, year);
        doy = encode(m, d, year);
    }
    void setYear(int y)    { year = y; }
    string getDate() const {
        Month m; int d;
        decode(doy, m, d, year);
        return to_string(m)+"/"+to_string(d)+"/"+to_string(year);
    }
};

int main() {
  DOB dob(Mar,14,2012), phd(Feb,23,1998);
  cout << dob.getDate() << " - " << phd.getDate() << endl;
  return 0;
}
```

MERRIMACK COLLEGE

Video: C++ Programming Tutorial - Encapsulation.

# Classes and Objects

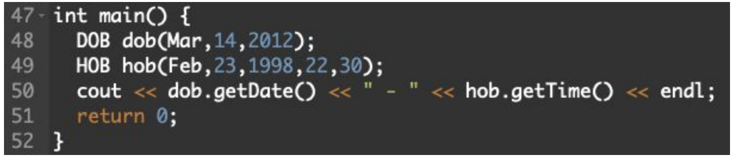
C++, once called C with Classes

Object-oriented concepts were the reason to create C++ language.

In C++, unlike Java, there are imperative portions and there is data structures encapsulated as objects.

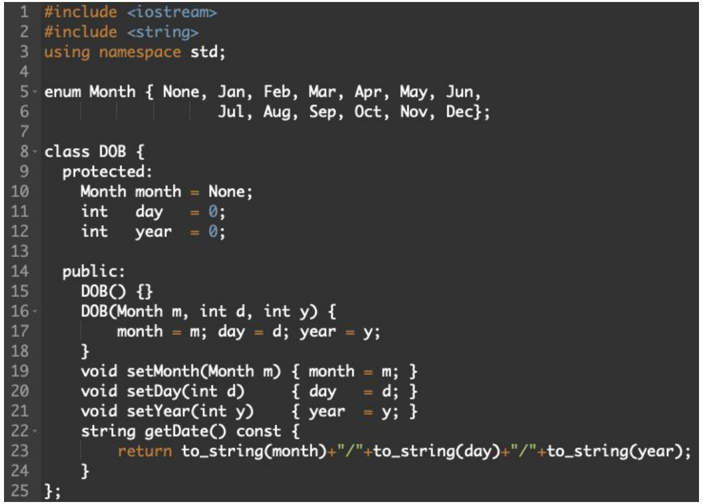
- From *struct* to *class*
- Methods
- Encapsulation
- ***Inheritance***
- Polymorphism

# Inheritance

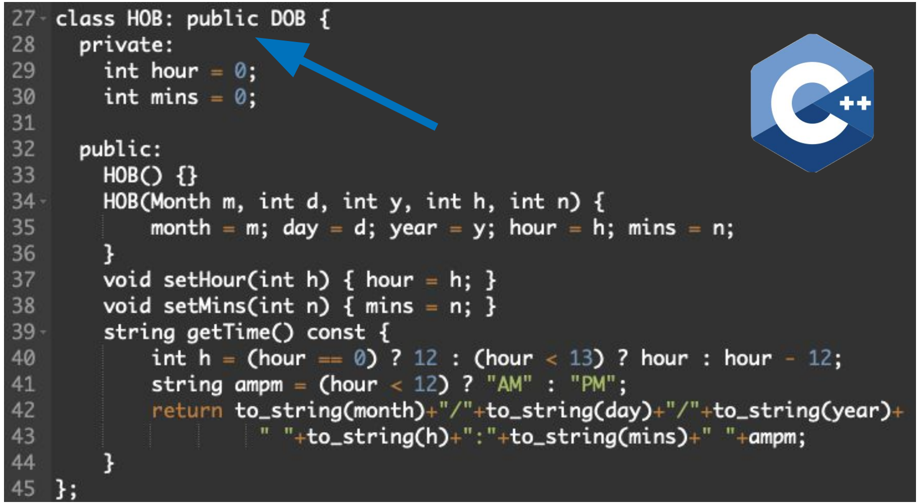


```
int main() {
    DOB dob(Mar,14,2012);
    HOB hob(Feb,23,1998,22,30);
    cout << dob.getDate() << " - " << hob.getTime() << endl;
    return 0;
}
```

The definition of subclasses in C++ is pretty similar to Python, since it is possible to declare a class as a **subclass** (sometimes referred as *derived* class) of a **superclass** (sometimes referred as *base* class).



```
#include <iostream>
#include <string>
using namespace std;

enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
                   Jul, Aug, Sep, Oct, Nov, Dec};

class DOB {
  protected:
    Month month = None;
    int   day   = 0;
    int   year  = 0;

  public:
    DOB() {}
    DOB(Month m, int d, int y) {
        month = m; day = d; year = y;
    }
    void setMonth(Month m) { month = m; }
    void setDay(int d)     { day   = d; }
    void setYear(int y)    { year  = y; }
    string getDate() const {
        return to_string(month)+"/"+to_string(day)+"/"+to_string(year);
    }
};
```

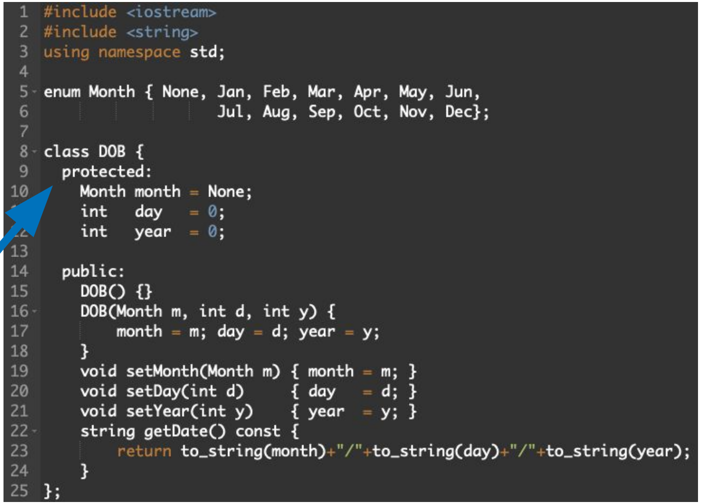


```
class HOB: public DOB {
  private:
    int hour = 0;
    int mins = 0;

  public:
    HOB() {}
    HOB(Month m, int d, int y, int h, int n) {
        month = m; day = d; year = y; hour = h; mins = n;
    }
    void setHour(int h) { hour = h; }
    void setMins(int n) { mins = n; }
    string getTime() const {
        int h = (hour == 0) ? 12 : (hour < 13) ? hour : hour - 12;
        string ampm = (hour < 12) ? "AM" : "PM";
        return to_string(month)+"/"+to_string(day)+"/"+to_string(year)+
                " "+to_string(h)+":"+to_string(mins)+" "+ampm;
    }
};
```

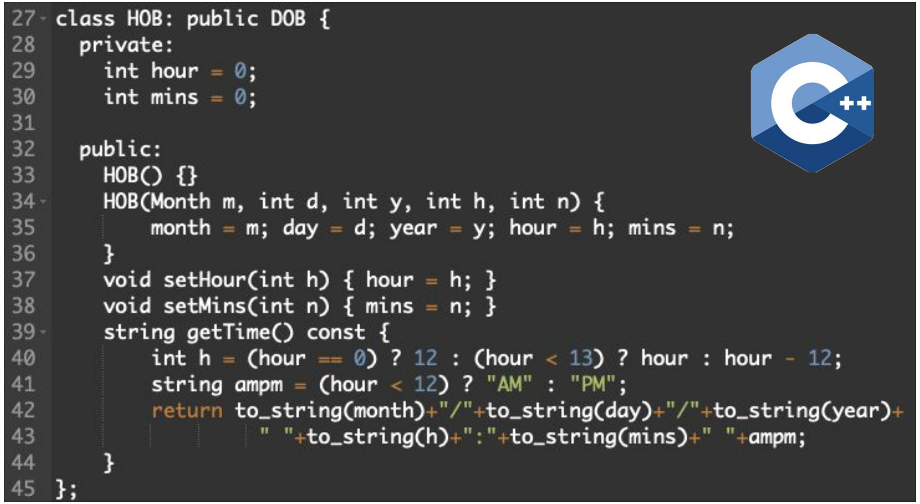
Video: C++ Programming Tutorial - Inheritance.

# Inheritance

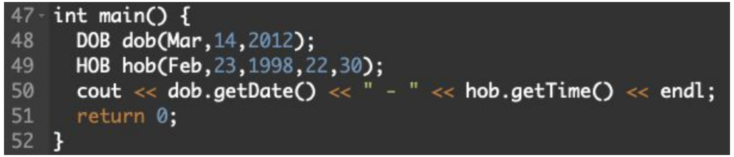
It is important to notice the use of ***protected*** declaration, as it allows the subclass to have access, while all others see it as ***private***.



```cpp
#include <iostream>
#include <string>
using namespace std;

enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
                   Jul, Aug, Sep, Oct, Nov, Dec};

class DOB {
  protected:
    Month month = None;
    int   day   = 0;
    int   year  = 0;

  public:
    DOB() {}
    DOB(Month m, int d, int y) {
        month = m; day = d; year = y;
    }
    void setMonth(Month m) { month = m; }
    void setDay(int d)     { day   = d; }
    void setYear(int y)    { year  = y; }
    string getDate() const {
        return to_string(month)+"/"+to_string(day)+"/"+to_string(year);
    }
};
```



```cpp
class HOB: public DOB {
  private:
    int hour = 0;
    int mins = 0;

  public:
    HOB() {}
    HOB(Month m, int d, int y, int h, int n) {
        month = m; day = d; year = y; hour = h; mins = n;
    }
    void setHour(int h) { hour = h; }
    void setMins(int n) { mins = n; }
    string getTime() const {
        int h = (hour == 0) ? 12 : (hour < 13) ? hour : hour - 12;
        string ampm = (hour < 12) ? "AM" : "PM";
        return to_string(month)+"/"+to_string(day)+"/"+to_string(year)+
                 " "+to_string(h)+":"+to_string(mins)+" "+ampm;
    }
};
```

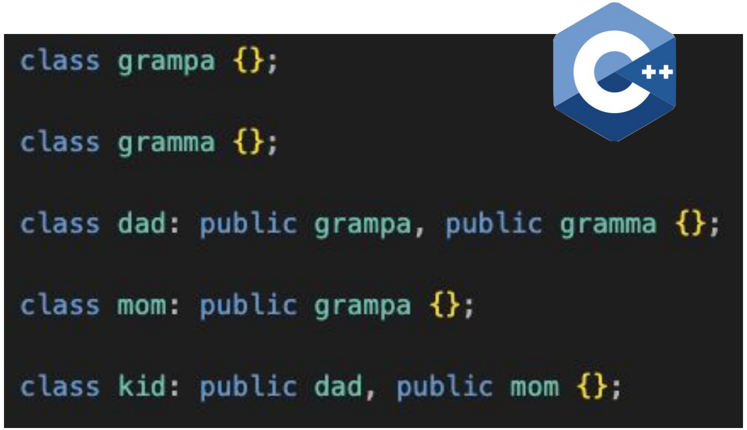


```cpp
int main() {
  DOB dob(Mar,14,2012);
  HOB hob(Feb,23,1998,22,30);
  cout << dob.getDate() << " - " << hob.getTime() << endl;
  return 0;
}
```

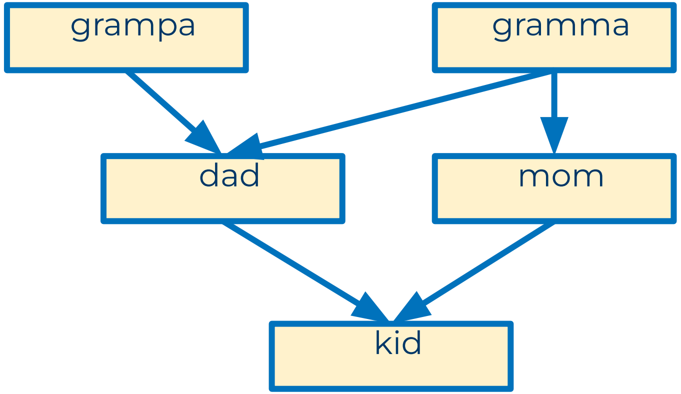
Text: C++ Inheritance.

# Inheritance

The definition of an inheritance graph is possible in C++, since it is possible to define  subclasses to multiple classes, as well as subclasses to subclasses.



```
class grampa {};

class gramma {};

class dad: public grampa, public gramma {};

class mom: public grampa {};

class kid: public dad, public mom {};
```



Video: C++ Base and Subclasses  Inheritance.

# Classes and Objects

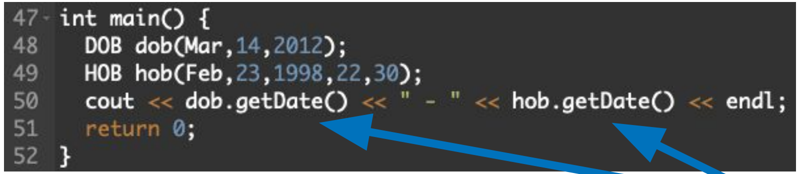
C++, once called C with Classes

Object-oriented concepts were the reason to create C++ language.

In C++, unlike Java, there are imperative portions and there is data structures encapsulated as objects.

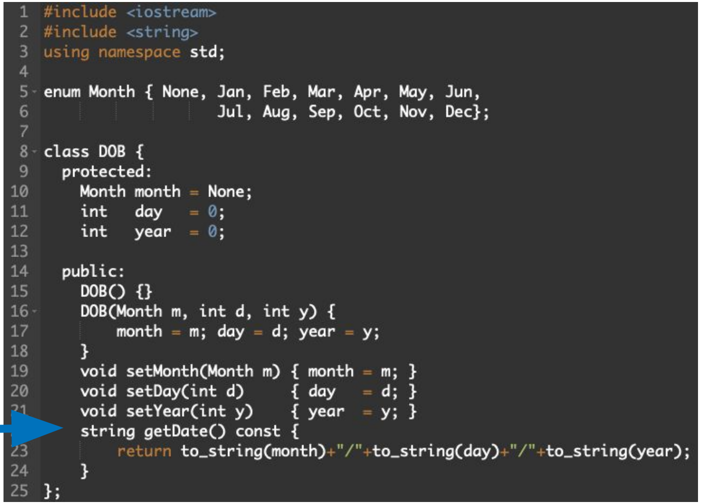
- From *struct* to *class*
- Methods
- Encapsulation
- Inheritance
- ***Polymorphism***

# Polymorphism

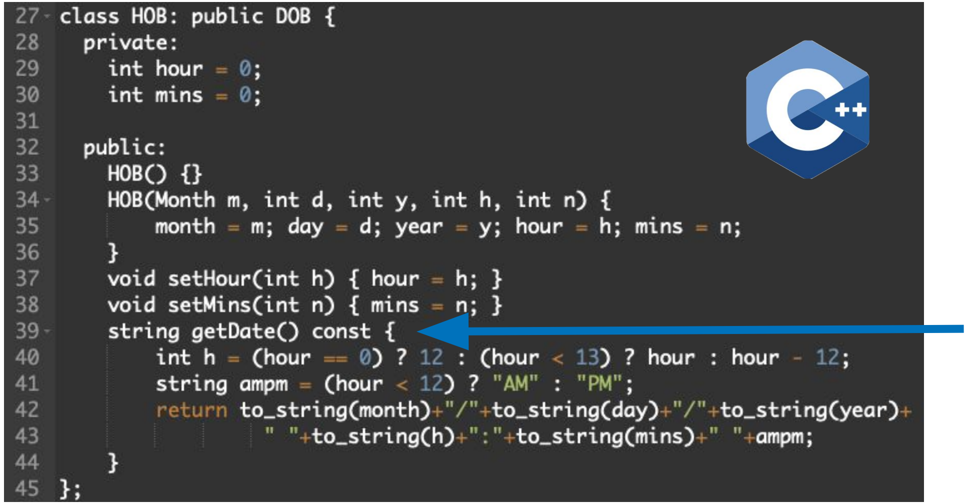


```
int main() {
  DOB dob(Mar,14,2012);
  HOB hob(Feb,23,1998,22,30);
  cout << dob.getDate() << " - " << hob.getDate() << endl;
  return 0;
}
```

The polymorphism in C++ is strongly related to the definition of related classes that define methods with the same name and set of parameters.



```
#include <iostream>
#include <string>
using namespace std;

enum Month { None, Jan, Feb, Mar, Apr, May, Jun,
                   Jul, Aug, Sep, Oct, Nov, Dec};

class DOB {
  protected:
    Month month = None;
    int   day   = 0;
    int   year  = 0;

  public:
    DOB() {}
    DOB(Month m, int d, int y) {
        month = m; day = d; year = y;
    }
    void setMonth(Month m) { month = m; }
    void setDay(int d)     { day   = d; }
    void setYear(int y)    { year  = y; }
    string getDate() const {
        return to_string(month)+"/"+to_string(day)+"/"+to_string(year);
    }
};
```



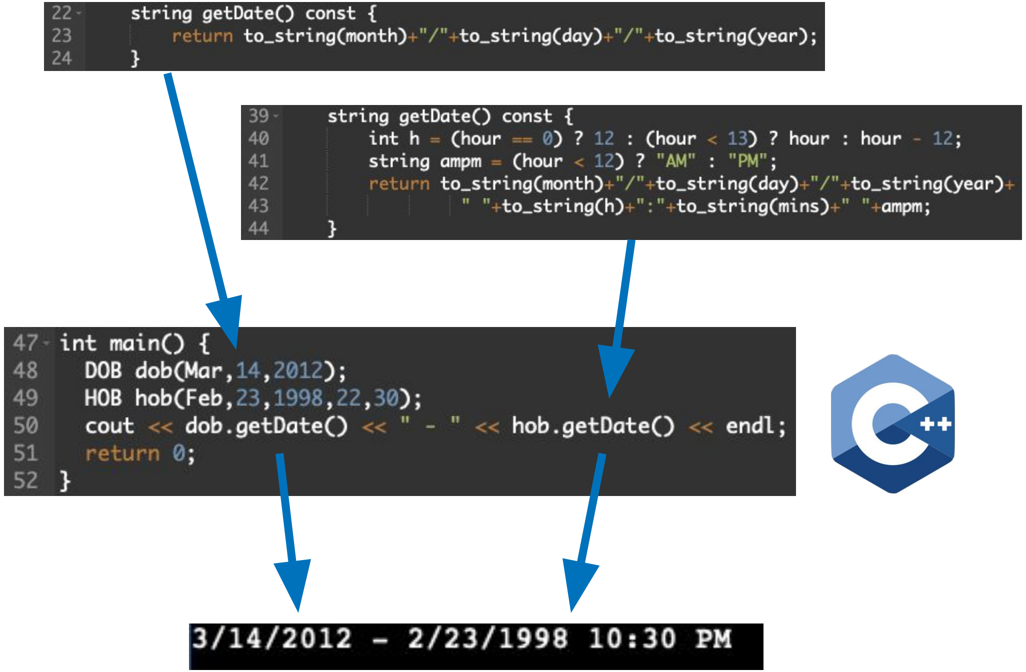
```
class HOB: public DOB {
  private:
    int hour = 0;
    int mins = 0;

  public:
    HOB() {}
    HOB(Month m, int d, int y, int h, int n) {
        month = m; day = d; year = y; hour = h; mins = n;
    }
    void setHour(int h) { hour = h; }
    void setMins(int n) { mins = n; }
    string getDate() const {
        int h = (hour == 0) ? 12 : (hour < 13) ? hour : hour - 12;
        string ampm = (hour < 12) ? "AM" : "PM";
        return to_string(month)+"/"+to_string(day)+"/"+to_string(year)+
                " "+to_string(h)+":"+to_string(mins)+" "+ampm;
    }
};
```

Text: C++ Polymorphism.

# Polymorphism

The polymorphism in C++ is strongly related to the definition of related classes that define methods with the same name and set of parameters.

An object calling such methods will decide which method to execute according to the classes of the object.



```
    string getDate() const {
        return to_string(month)+"/"+to_string(day)+"/"+to_string(year);
    }
```

```
    string getDate() const {
        int h = (hour == 0) ? 12 : (hour < 13) ? hour : hour - 12;
        string ampm = (hour < 12) ? "AM" : "PM";
        return to_string(month)+"/"+to_string(day)+"/"+to_string(year)+
                " "+to_string(h)+":"+to_string(mins)+" "+ampm;
    }
```

```
int main() {
  DOB dob(Mar,14,2012);
  HOB hob(Feb,23,1998,22,30);
  cout << dob.getDate() << " - " << hob.getDate() << endl;
  return 0;
}
```

```
3/14/2012 - 2/23/1998 10:30 PM
```

Video: C++ Programming Tutorial Polymorphism.

# This Week's tasks

- In-class Exercise E#2
- Coding Project P#2
- Quiz Q#2

Tasks

- Execute examples shown in class.
- Create your own C++ program twitch a class for square matrices.
- Quiz #2 about this week topics.

# In-class Exercise - E#2

Execute examples shown in class. Namely:

- Example of use of struct (slide 7);
- Example of class definition (slide 10);
- Example of overloading (slide 13);
- Example of encapsulation (slides 18-19-20);
- Example of inheritance and polymorphism (slide 26).

To all examples, you have to reproduce the presented code and run it successfully. Take a printscreen of each one of the examples' output.

You have to submit a ***.pdf*** file with the output for the five examples above.

This task counts towards the In-class Exercises grade and the deadline is This Saturday.

# Second Coding Project - P#2

Create your own C++ program with the definition of a class ***squareMatrix*** (same number of rows and columns) that holds the order (the number of rows and columns) and the elements of the square matrix (Real numbers stored in a bidimensional array). The class should have at least:

- One constructor parameterized with the matrix order;
- One getter method that receives the row and column indices and delivers the element;
- One setter method that receives the the row and column indices, plus the value of the element to assign;
- One getter method that delivers in a single dimension vector the diagonal elements of the matrix.

Create a **main** function with an example matrix and call for all methods.

Your task: Go to Canvas, and submit your C++ file (.cpp) within the deadline.

This assignment counts towards the Projects grade and the deadline is Next Tuesday.

# Second Quiz - Q#2

- The second quiz in this course covers the topics of Week 2;
- The quiz will be available this Saturday, and it is composed by 10 questions;
- The quiz should be taken on Canvas (Module 2), and it is not a timed quiz:
    - You can take as long as you want to answer it;
- The quiz is open book, open notes, and you can even use any language Interpreter to answer it;
- Yet, the quiz is evaluated and you are allowed to submit it only once.

Your task:

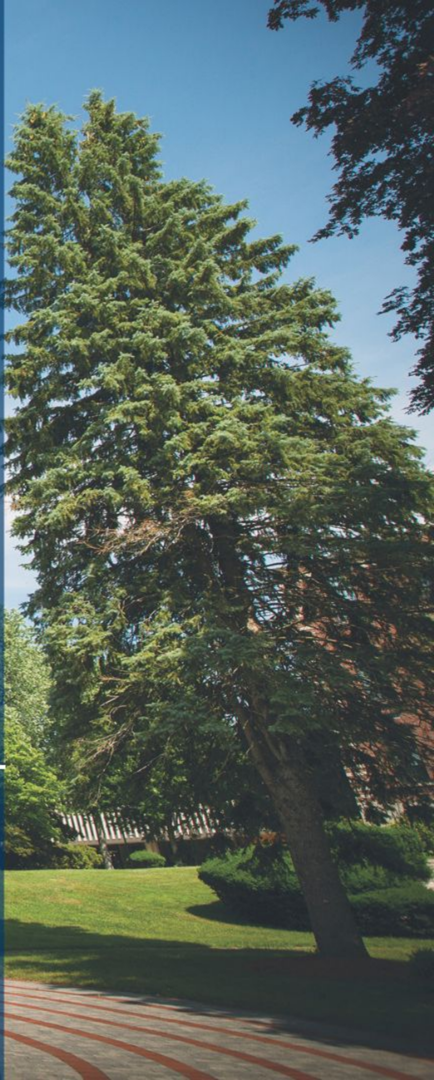
- Go to Canvas, answer the quiz and submit it within the deadline.

This quiz counts towards the Quizzes grade and the deadline is Next Tuesday.

## Week 2 of CSC 6033

- Do In-class Exercise E#2 until Saturday;
- Do Quiz Q#2 (available Saturday) until next Tuesday;
- Do Coding Project P#2 until next Tuesday.

Next Week - Finite Automata



MERRIMACK COLLEGE