Example – struct Version of Date "Object"

```c
struct Datatype
{
    int day;
    int month;
    int year;
};
```

*structs are part of C and C++
Members of structs are data values*

Class Version of Date "Object"

```c
class Dateclass
{
    public:
    int day;
    int month;
    int year;
    void outdate();
};
```

*We can now include "methods" (functions)
in the declaration as well as detail*

*NOTE: classes are only available in C++ (not C)*
(In fact, C++ stands for "C plus Classes")
Using the Date Class

#include "dateclass.h"

void main()
{
    Dateclass today;
    .
    .
    today.month = 2;
    today.day = 27;
    today.year = 1963;
    today.outdate();
} // End main

Methods are referenced just like other members

The outdate() Method

void Dateclass::outdate(void)
{
    cout << month << '/' << day << '/'
         << year << endl;
} // END outdate

Other members of the class are available to all
the methods of the class.
Information Hiding Version of Dateclass

```cpp
class Dateclass
{
    public:
        void outdated();
        int setdate(int, int, int);
    private:
        int day;
        int month;
        int year;
};
```

Now, the two methods (outdate and setdate) are available to the "outside world," but the data members are not.

By default, ALL class members are private unless specified otherwise.

Using the Date Class

```cpp
#include "dateclass.h"

void main()
{  
    Dateclass today;
    today.month = 2; // no longer legal!!
    today.setdate(2, 27, 2001);
    today.outdate();
} // End main
```
Classes and Source Files

Source (code) files for classes can be organized in many ways. The following is a typical organization:

File 1 - dateclass.h - contains only the class declaration. Sometimes called the "Interface" for the class.

File 2 - dateclass.c - contains the definitions for all methods (functions) in the class.

File 3 - xxxxxx.c - contains the main program and all (non-class) functions.

Typical File Organization for Classes

```
exampclass.h

```class exampclass
{
    public:
        void method1(int i)
        {
            // code for method1 goes here
        } // END method1

    int method2(char ch)
    {
        // code for method2 goes here
        return val;
    } // END method2

} // END class exampclass
```

```
exampclass.cpp

```#include "exampclass.h"

void exampclass::method1(int i)
{
    // code for method1 goes here
} // END method1

int exampclass::method2(char ch)
{
    // code for method2 goes here
    return val;
} // END method2
```

```
main.cpp

```#include "exampclass.h"

int main()
{
    exampclass x, y;
    x.method1();
    y.method2(ch);
    ...
    return 0;
} // END main
```

- Called the header or interface file
- Only contains class member declarations
- Contains method definitions
- Contains method instantiations
Compiling classes - Examples

To compile everything at once
```
g++ main.cpp examplclass.cpp
```

Separate compilation of class methods, then link with main
```
g++ -o examplclass.cpp
```  
```
g++ main.o examplclass.o
```

Separate compilation of main and class, then create an executable named "example"
```
g++ -c examplclass.cpp
```  
```
g++ -o main.cpp
```  
```
g++ -o example main.o examplclass.o
```

Dateclass with Constructor

class Dateclass
{
    public:
        void outdate();
        int setdate(int, int, int);
        Dateclass();
    private:
        int day;
        int month;
        int year;
};

constructor method
Constructor Definition

```cpp
Dateclass::Dateclass
{
    day = 1;
    month = 1;
    year = 2001;
} // END Dateclass constructor
```

*Note: no type*

*The constructor is like a function, except that it doesn't have a type.*

---

Overloaded Constructors

```cpp
class Dateclass
{
    public:
        void outdate();
    int setdate(int, int, int);
    Dateclass();
    Dateclass(int, int, int);
    private:
        int day;
        int month;
        int year;
};
```
Use of Constructors

```c
int main()
{
    Dateclass a(1,4,2001);

    Dateclass b;  // Uses 3-argument constructor

    .
    .
    .  // Uses "default" constructor (with no arguments)
```