Object-ive Software Coding Standards

1. Constructors should always be written for the creation and initialization of objects. At the very minimum, a default constructor should be written to initialize all data members.

```
// File:  date.h
class Date{
public:
  Date(int m, int d, int y);
private:
  int day;
  int month;
  int year;
};

// File:  date.cpp
Date::Date(int mo, int dy, int yr) : day(dy), month(mo), year(yr)
{
}
```

2. Avoid using the C language I/O features (printf, scanf, getchar). Instead use the C++ I/O objects: cin and cout.

```
float somevar;
cout << "somevar=" << somevar << endl;
```

3. Use common sense commenting. If it isn't obvious what a segment of code is going to do or what the variable is for, add a comment explaining its purpose.

4. Initialization lists should be used in constructors as much as possible. Even built-in types should use the initialization list.

```
// File:  person.h
#include <string>
class Person{
public:
  Person(string n, int a);
private:
  int age;
  string name;
};

// File:  person.cpp
#include "person.h"
Person::Person(string nm, int a) : age(a), name(nm)
{
}
```

5. Use proper data encapsulation. Member data should always be either protected or private.

6. The class declaration of each class should be placed in a header file with the same name as the class. The definition of the class's member functions should be placed in a separate source file named the same as the class:

```
//File:  date.h
class Date {
public:
  void display();
  ...
private:
  int day;
  ...
};

//File:  date.cpp
void Date::display()
{
  cout << month << '/' << day << '/' << year << endl;
}
```

7. All header files should start and end with preprocessor directives to prevent multiple #include problems. By convention, the symbol names should be the same as the filename, using an underscore instead of a period. Also, the symbol should be all-uppercase:

```
//File:  date.h
#ifndef DATE_H
#define DATE_H
class Date {
...
};
#endif
```

8. Whenever new is used to create an object, delete should be used to release it.

```
Date * dp;
dp = new Date(7,14,1776);
dp->display();
...
delete dp;
```

9. The destructor should always be virtual if there are any virtual functions defined in a class.

```
class Date {
public:
  virtual void display();
  virtual ~Date();
};
```

10. Any file that is opened should be explicitly closed:

```
ifstream inf("data.file");
...
inf.close();
```

11. Constants and references:

    (a) Any non-builtin type (e.g. "string", "Date", "Book") passed as an argument to a function should be passed by reference or by pointer:

    ```
    void func1(Date& dt);
    void func2(Date* dt);
    ```

    (b) Returning from a function: Non-builtin types (e.g. "string", "Date", "Book") *which are not local to the function* should be returned by reference or by pointer.

    ```
    Date& func1(void);
    Date* func2(void);
    ```

    (c) Any reference or pointer parameter or return type that won't be modified should be specified as const:

    ```
    const Date& func1(const Date& dt);
    const Date* func2(const Date* dt);
    ```

    (d) Any member function that is not designed to change its member data should be made a constant member function:

    ```
    void Date::func1(void) const;
    ```