Using the following part of a class, implement the `Sharpen()` method. It removes 1 from the length:

```cpp
class Pencil
{
public:
    bool Sharpen();
private:
    int m_length;
};
```

What's going on here?
Constructors

Special Methods that “build” (construct) an object
Supply default values
Initialize an object

Syntax:
ClassName();
ClassName::ClassName(){ /* code */ }

Notice
No return type
Same name as class!

Constructor Example

class DayOfYear
{
    public:
    DayOfYear( int initMonth, int initDay );
    void Input();
    void Output();
    void Set( int newMonth, int newDay );
    void Set( int newMonth );
    int GetMonthNumber();
    int GetDay();
    private:
    int m_month;
    int m_day;
};

Constructor Example Implementation

DayOfYear::DayOfYear( int initMonth, int initDay )
{
    m_month = initMonth;
    m_day = initDay;
}

    // Improved version
    DayOfYear::DayOfYear( int initMonth, int initDay )
    {
        Set( initMonth, initDay );
    }
Constructor Example Implementation

Initialization Lists
   Alternative to assignment statements (sometimes necessary!)
   Comma-separated list following colon in method definition
Syntax:
   DayOfYear::DayOfYear( int initMonth, int initDay )
     : m_month( initMonth ), m_day( initDay )
   {
   }

Overloading Functions

C limitation
   Functions are unique based on name
C++ extension
   Functions are unique based on name AND parameter list (type and number)
Overloading
   Declaring two or more functions with same name
   Must have different parameter lists
   Return types are NOT used to differentiate functions

Overloading Constructors

Yes – different parameter lists
Example
   class DayOfYear
   {
     public:
       DayOfYear( int initMonth, int initDay );
       DayOfYear( int initMonth );
       DayOfYear( );
       // other public methods.
     private:
       int m_month;
       int m_day;
   };
Overloading Example

```c++
int AddTwo(int a, int b) {
    return a + b;
}

double AddTwo(double a, double b) {
    return a + b;
}

int main() {
    cout << AddTwo(3, 4) << endl;
    cout << AddTwo(3.0, 4.0) << endl;
    cout << AddTwo(3, 4.0) << endl;
    cout << AddTwo(3.0, 4) << endl;
    return 0;
}
```

Interesting...

What happens with this?

```c++
int AddTwo(int a, int b) {
    return a + b;
}

double AddTwo(double a, double b) {
    return a + b;
}

int main() {
    cout << AddTwo(3.0, 4) << endl;
    return 0;
}
```

Overloading Constructors

```c++
DayOfYear::DayOfYear(int initMonth, int initDay) {
    Set(initMonth, initDay);
}

DayOfYear::DayOfYear(int initMonth) {
    Set(initMonth, 1);
}

DayOfYear::DayOfYear() {
    Set(1, 1);
}
```
Overloading Constructors

class DayOfYear
{
    public:
        DayOfYear( int initMonth = 1, int initDay = 1 );
        // other public methods.
    private:
        int m_month;
        int m_day;
};

DayOfYear::DayOfYear( int initMonth, int initDay )
{
    Set(initMonth, initDay);
}

Constructors

Why haven't we seen this before?
    Compiler builds a default constructor
        Unless you define a constructor...
Think about the following:
    DayOfYear bachBirthday;
        Calls default constructor for DayOfYear!
What if something goes wrong?
    Throw exception (later...)

Const and Objects

With an Object

    const DayOfYear jan1st(1, 1);
    jan1st.Set(1, 5);  // ERROR

myfile.cpp: In function 'int main()':
myfile.cpp:20: passing 'const DayOfYear' as
    'this' argument of 'void DayOfYear::Set(int, int)' discards qualifiers
Const and Methods

Const member functions
  Promise not to modify the current object
  Usually accessors, print functions, ...

Compiler checks
  Directly – is there an assignment to data member in method?
  Indirectly – is there a call to a non-const method?

Syntax
  retType methodName(parameters) const;

Const Example

class DayOfYear
{
  public:
    DayOfYear( int initMonth = 1, int initDay = 1 );
    void Input( );
    void Output( ) const;
    void Set( int newMonth, int newDay );
    void Set( int newMonth );
    int GetMonthNumber( ) const;
    int GetDay( ) const;
  private:
    int m_month;
    int m_day;
};

Const Rules

Const member functions
  Can be called on const and non-const objects
  Can call other const member functions
  Cannot call non-const member functions

Non-const member functions
  Can be called only on non-const objects
  Otherwise, compiler error!
  Can call const and non-const member functions

Const objects
  Can be passed as const argument
Non-const objects
  Can be passed as const or non-const argument
Practice
What is wrong with this?

```cpp
int DayOfYear::GetDay ( ) const 
{
    if (m_day < 1 )
        Set ( m_month, 1 );
    return m_day;
}
```

Practice
What is wrong with this?

```cpp
void Bob ( const DayOfYear& doy)
{
    OutputDayOfYear ( doy );
    cout << "Please enter your birth month and day \n";
    int birthMonth, birthDay;
    cin >> birthMonth >> birthDay;
    doy.Set( birthMonth, birthDay );
}
```

Implementing with Const
Start from the beginning
Don't try to add const at the end of implementing
Use for
Member functions that don't change object
Facilitators (maybe) and Accessors (most definitely)
Parameters whenever reasonable
Not with pass-by-value
Yes with pass-by-reference
**Aggregation**

Objects can hold other objects!

Class defines a private data member of another Class-type

"has-a" relationship

Example

```cpp
class Student
{
public:
    // some methods...
private:
    Address m_address;
    // more data.
};
```

**Aggregation – Another Look**

```cpp
class Vacation
{
public:
    Vacation(int month, int day, int nbrOfDays);
    // more methods...
private:
    DayOfYear m_startDay;
    int m_lengthOfTrip;
    // more data.
};
Vacation::Vacation(int month, int day, int nbrOfDays)
    : m_startDay(month, day), m_lengthOfTrip(nbrOfDays)
{
    // code...
}
```

What's going on here?

Implicitly call the Constructor!

Remember – initializer list is the only way to call Constructor!

Can Vacation access DayOfYear's private data members?
Aggregation

House "has-a"
FrontDoor
Set of bedrooms
Garage
Address
Garage "has-a"
Lawnmower
Rake
Car
Car "has-a"
Driver
Set of passengers
Driver "has-a"
Name
Address
...

You can have as many layers of aggregation as you need – until you get to a set of primitive types!

Static

int foobar()
{
    int a = 10;
    ++a;
    return a;
}

int foobar()
{
    static int a = 10;
    ++a;
    return a;
}

What is returned?

Ah…tricky…

'a' retains its value between calls to foobar…

11, 11, 11, 11, 11, …

Static and Classes

Static data member
ALL objects share data
If one changes, affects all
Static methods
Can access static data
CANNOT access non-static data or methods
Regular methods
Can access static data
Can access non-static data and methods
**Static Example**

```cpp
class Person
{
public:
    static bool SpendMoney(int amount);
private:
    static Wallet m_wallet;
    Wallet m_moneyClip;
};
// In Person.h
Wallet Person::m_wallet(0);
bool Person::SpendMoney(int amount)
{
    m_wallet.RemoveMoney(amount);
    m_moneyClip.RemoveMoney(amount); // compiler error!!!
}

// In main
// Create a person
Person Bob;
// Bob adds money to the wallet
Bob.AddMoney(100);
// Anyone can call SpendMoney!
Person::SpendMoney(100);
// Bob has no money!
Bob.SpendMoney(10); // fails!!
If any money is spent, everyone has lost that money!
```

**Designing Classes**

Ask yourself the following questions:
- What are the responsibilities of this type of object?
- What actions can an object take?
- What actions can another function take on an object?
- What information does an object store?
- What information does an object need access to?

For each method:
- What parameters (const, ref, const-ref, val)?
  - Preconditions – what values are legal for parameters?
- What return value (const, ref, const-ref, val)?
  - Postconditions – what was altered by method?
- Does this method change the object (const, non-const)?

**Incremental / Modular Development & Compilation**

*General Programming Approach*

*Bottom-Up Development*
- Work on one class
- Write one method at a time
- Develop, test, repeat
- Test class in isolation

*Bottom-Up Testing*
- Test one class in isolation
- Test two classes in isolation (when they are connected)
- Test all classes together