Chapter - 18
Operator Overloading
Fixed Point Numbers

In floating point the decimal point can appear anywhere.

0.23 123.0 1.234

In fixed point the number of digits after the decimal point is a fixed number (for example 2)

12.00 123.00 0.01 45.83 0.33
namespace fixed_pt {

Basic Fixed Point Number Class
const int fixed_exp = 100;
  // 10**fixed_point */

class fixed_pt {
  {
    private:
      // Value of our fixed point number
      long int int value;

      // ....
  };

Numbers are stored as integers (value = number * fixed_point_exp).
// Default constructor, zero everything
fixed_pt() : value(0) {
}

// Copy constructor
fixed_pt(const fixed_pt& other_fixed_pt) :
  value(other_fixed_pt.value) {
}

// Construct a fixed_pt out of a double
fixed_pt(const double init_real) :
  value(double_to_fp(init_real)) {
}
Converting a double to a fixed point

Basic Function

```cpp
fixed_pt(double init_real) {
    value = init_real *
        fixed_exp);
}
```

But:
1. Some casts are missing
2. Testing has shown that due to floating point errors, we need a fudge factor. (Chapter 19 will detail what floating point can do to you.)

Actual conversion function:

```cpp
const double fixed_fudge_factor = 0.0001;

static long int double_to_fp(
    const double the_double) {
    return (static_cast<long int>(
        the_double *
            static_cast<double>(fixed_exp) +
            fixed_fudge_factor));
}
```
Adding two fixed point numbers

```cpp
inline fixed_pt add(
    const fixed_pt& oper1,
    const fixed_pt& oper2
) {
    fixed_pt result.value =
        oper1.value + oper2.value;
}
```

Usage:

```cpp
fixed_pt i1(12.34), i2(45.67);
fixed_pt i3 = add(i1, i2);
```
Using the "operator" functions

```cpp
inline fixed_pt operator+(const fixed_pt& oper1, const fixed_pt& oper2) {
    fixed_pt result.value = oper1.value + oper2.value;
}

Usage:

    fixed_pt i1(12.34), i2(45.67);
    fixed_pt i3 = operator+(i1, i2);

Or:

    fixed_pt i3 = i1 + i2;
```
Binary Operators

+  Addition
-  Subtraction
*  Multiplication
/  Division
%  Modulus
^  Bitwise exclusive OR
&  Bitwise and
|  Bitwise or
<< Left Shift
>> Right Shift
Relational Operators

==   Equality          !=   Inequality
<    Less Than        <=   Less or equal
>    Greater          >=   Greater or equal

```cpp
inline bool operator == (
    const fixed_pt& oper1,
    const fixed_pt& oper2
) {
    return (oper1.value ==
            oper2.value);
}
```
Unary Operators

+    Positive       -    Negative
*    Dereference   &    Address of
~    One's complement (invert bits)

inline fixed_pt operator - (  
    const fixed_pt& oper1
) {
    return (  
        fixed_pt(-oper1.value);
    }

Shortcut Operators

+=      Increase
*+      Multiply by
%=      Remainder
&=      And into
<<=     Shift left
-=      Decrease
/=      Divide by
^=      Exclusive Or into
|=      Or into
>>=     Shift right

inline fixed_pt& operator += (  
    fixed_pt oper1,
    const fixed_pt& oper2
) {  
    oper1.value += oper2.value;
    return (oper1);
}
Increment Operators

Increment comes in two forms

++i: Increment, then return value
i++: Increment, return value before increment.

A dummy parameter is used to distinguish between the two.
Fixed Point ++

// Prefix x = ++f
inline fixed_pt& operator ++(fixed_pt& oper) {
    oper.value += fixed_point_exp;
    return (oper);
}

// Postfix x = f++
inline fixed_pt operator ++(fixed_pt oper, int) {
    fixed_pt result(oper);  // Save return
    oper.value += fixed_point_exp;
    return (result);
}
// NOTE THE RETURN TYPE DIFFERENCE IN THIS FUNCTION
Output Operator

```cpp
inline std::ostream& operator << (std::ostream& out_file, const fixed_pt& number) {
    long int before_dp = number.value / fixed_exp;
    long int after_dp1 = abs(number.value % fixed_exp);

    long int after_dp2 = after_dp1 % 10;

    after_dp1 /= 10;

    out_file << before_dp << '.' << after_dp1 << after_dp2;
    return (out_file);
}
```
Input Operator

```cpp
inline std::istream& operator >> ( 
std::istream& in_file, fixed_pt& number ) {
    char before_dp;
    char dot;
    char after_dp1, after_dp2;

    in_file >> before_dp >> dot >>
    after_dp1 >> after_dp2;

    number.value = before_dp * fixed_exp +
                  (after_dp1 - '0') * 10 +
                  (after_dp2 - '0');
}
```

It is not this simple
Input sentry – error marking class

At the beginning of the input function

```cpp
std::istream::sentry
    the_sentry(in_file, true);
```

The "true" tell the sentry to skip any leading whitespace in the text.

Check the sentry:

```cpp
if (the_sentry) {
    // .. Read the file
} else {
    in_file.setstate(
        std::ios::failbit);
}
```
Starting the read

```cpp
in_file >> before_dp;
if(in_file.bad()) return (in_file);

in_file >> dot_ch;
if(in_file.bad()) return (in_file);

if (dot_ch != '.') {
    in_file.setstate(
        std::ios::failbit);
    return (in_file);
}
// continue with
//... read ... error check ... fail
```
Operator functions as members

Operator member functions are the same as non-member functions except the first argument is an implied this.
Operator functions as members

```cpp
inline fixed_pt operator + (const fixed_pt& n1, const fixed_pt& n2) {
    fixed_pt result(n1.value + n2.value);
    return (result);
}

class fixed_pt {
    //...
    public:
    fixed_pt operator + (const fixed_pt& n2) const
    fixed_pt result(
        value + n2.value);
    return (result);
}
```
What's wrong with this program?

When run it prints?
   Copy constructor called
   Copy constructor called
   .. continues forever.