**Overloading Input and Output Stream Operators**

**3rd class – PhoneNumber.h:**

#ifndef PHONENUMBER\_H

#define PHONENUMBER\_H

#include <iostream>

#include <string>

class PhoneNumber{

friend std::ostream& operator<<(std::ostream&, const PhoneNumber&);

friend std::istream& operator>>(std::istream&, PhoneNumber&);

private:

std::string areaCode;

std::string exchange;

std::string line;

};

#endif

**2nd class – PhoneNumber.cpp:**

#include "stdafx.h"

#include <iomanip>

#include "PhoneNumber.h"

using namespace std;

ostream& operator<<(ostream& output, const PhoneNumber& number) {

output << "Area code: " << number.areaCode << "\nExchange: "

<< number.exchange << "\nLine: " << number.line << "\n"

<< "(" << number.areaCode << ") " << number.exchange << "-"

<< number.line << "\n";

return output; // enables " cout << a << b << c ";

}

istream& operator>>(istream& input, PhoneNumber& number) {

input.ignore(); // skip (

input >> setw(3) >> number.areaCode;

input.ignore(2); // skip ) and space

input >> setw(3) >> number.exchange; // input exchange

input.ignore(); // skip dash (-)

input >> setw(4) >> number.line; // input line

return input;

}

**1st class – ConsoleApplication3.cpp:**

#include "stdafx.h"

#include <iostream>

#include <string>

#include "PhoneNumber.h"

int main()

{

PhoneNumber phone;

int a;

std::cout << "Enter phone number in te form (555) 555-5555:\n";

/\*

cin >> phone invokes operator>> by implicitly issuing

the non-member function call operator>>(cin, phone)

\*/

std::cin >> phone;

std::cout << "The phone number entered was:\n";

/\*

cout << phone invokes operator<< by implicitly issuing

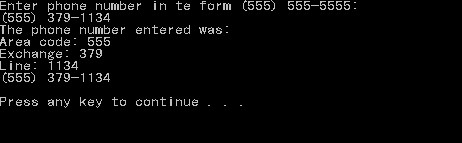
non-member function call operator<<(cout, phone)

\*/

std::cout << phone << std::endl;

}

**Result:**



**Important notes:**

* We just overloaded a binary operator
* (Stated in book) A binary operator can be overloaded as a non-static member function with one or as a non-member function with two parameters (one of those parameters must be either a class object or a reference to a class object). A non-member operator function is often declared as friend of a class for performance reasons.
* (Stated in book) Overloaded operator functions for binary operators can be member functions only when the left operand is an object of the class in which the function is a member. Example:

class String {

public:

bool operator<(const String&) const;

...

};