

```
#include <iostream>
#include <iterator>
#include <numeric>
#include <fstream>
#include <string>
#include <map>

using namespace std;

typedef std::map<int, string> roman_map;

string latin_to_roman(int lat_num);

void handle_io(int latin_number)
{
    cout << latin_number << " = "
         << latin_to_roman(latin_number)
         << endl;
}
```

```
int main(int argc, char* argv[])
{
    ifstream* infile = 0;
    istream* in = &cin;

    if (argc == 2)
    {
        infile = new ifstream(argv[1]);

        if ( ! *infile )
        {
            cerr << "File '" << argv[1]
                  << "' could not be opened."
                  << endl;
        }
        else
        {
            in = infile;
        }
    }

    istream_iterator<int> isi(*in);
    istream_iterator<int> eof;

    for_each(isi, eof, handle_io);

    delete infile;

    return 0;
}
```

```
class Int_Iterator
{
public:
    explicit Int_Iterator(int start = 0)
        : current(start) {}

    Int_Iterator& operator++()
    {
        return --current, *this;
    }

    Int_Iterator operator++(int)
    {
        Int_Iterator save(*this);
        ++*this;
        return save;
    }
    int operator*()
    {
        return current;
    }

    bool operator==(Int_Iterator& rhs)
    {
        return current == rhs.current;
    }

    bool operator!=(Int_Iterator& rhs)
    {
        return !(*this == rhs);
    }

private:
    int current;
};
```

```
class Concatenate
{
public:
    Concatenate(string const& s, string& r)
        : value(s), result(r) {}

    int operator()(int)
    {
        result += value;
        return 0;
    }

private:
    string const& value;
    string& result;
};

string operator*(string const& a, int n)
{
    string result;
    for_each(Int_Iterator(n),
              Int_Iterator(),
              Concatenate(a, result));
    return result;
}
```

```
class Convert
{
public:
    Convert(string& out, int n)
        : rom_str(out), lat_num(n) {}

    void
    operator()(pair<int, string> const& digit)
    {
        string glyph = digit.second;
        int value = digit.first;

        if (lat_num >= value)
        {
            rom_str += glyph * (lat_num / value);
            lat_num %= value;
        }
    }
private:
    string& rom_str;
    int lat_num;
};
```

```
string latin_to_roman(int lat_num)
{
    roman_map translation;

    translation[1] = "I";
    translation[5] = "V";
    translation[9] = "IX";
    translation[10] = "X";
    translation[50] = "L";
    translation[90] = "XL";
    translation[100] = "C";
    translation[500] = "D";
    translation[900] = "CM";
    translation[1000] = "M";

    string rom_str;

    for_each(translation.rbegin(),
              translation.rend(),
              Convert(rom_str, lat_num));

    return rom_str;
}
```