**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 2**

Тема: Перегрузка операторов в С++

Студент: Шараковский Юрий Дмитриевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Вариант 18: Создать класс IPAddress. Класс состоит из четырех чисел unsigned char. Реализовать арифметические операции сложения, вычитания, а также операции сравнения, так же реализовать функцию, которая будет определять принадлежность адреса к подсети по адресу подсети и битовой маске подсети. Операции сложения, вычитания и сравнения необходимо реализовать в виде перегрузки операторов. Необходимо реализовать пользовательский литерал для работы с константами типа IPAddress.

1. Описание программы

Программа описывает класс IPAddress соответствующий варианту задания. Программа позволяет вводить объекты этого класса со стандартного потока ввода и выполнять над ними стандартные операции (арифметические, операции сравнения, проверка принадлежания адреса подсети). В программе приведен пример создания и использования пользовательского литерала класса IPAddress.

1. Набор тестов

testcase01

В этом тесте проверяется базовая функциональность программы и то как она справляется с ошибками ввода или некорректно введенными данными, проверяется возможность определить то, принадлежит ли адрес подсети по адресу подсети и маски подсети.

calc 000.000.000.000 0.0.0.0

calc 000.000.000.000 123.456.789.012

subnet 192.168.1.30 192.168.0.0 255.255.0.0

calc -1 f

exit

testcase02

В этом тесте проверяется корректная работа операторов сравнения и операторов логического И, ИЛИ, НЕ, исключающего ИЛИ.

calc 100.50.100.100 100.50.100.100

calc 100.50.100.100 100.100.50.100

calc 123.123.123.123 123.123.123.123

calc 100.100.100.50 100.50.100.100

calc 255.255.85.85 255.255.170.170

exit

1. Результаты выполнения тестов

testcase01

$ ./oop\_exercise\_02

enter 'exit' to quit program, enter 'calc' <ADDR1> <ADDR2>

or 'subnet' <ADDR> <NET> <MASK> to check for subnet

addresses are entered in 123.123.123.123 format

calc 000.000.000.000 0.0.0.0

ADDR1: 000.000.000.000 ADDR2: 000.000.000.000

ADDR1 - ADDR2: 000.000.000.000

ADDR1 + ADDR2: 000.000.000.000

ADDR1 < ADDR2: 0

ADDR1 > ADDR2: 0

ADDR1 = ADDR2: 1

ADDR1 & ADDR2: 000.000.000.000

ADDR1 | ADDR2: 000.000.000.000

~ADDR1: 255.255.255.255

calc 000.000.000.000 123.456.789.012

ADDR1: 000.000.000.000 ADDR2: 123.200.021.012

ADDR1 - ADDR2: 133.056.235.244

ADDR1 + ADDR2: 123.200.021.012

ADDR1 < ADDR2: 1

ADDR1 > ADDR2: 0

ADDR1 = ADDR2: 0

ADDR1 & ADDR2: 000.000.000.000

ADDR1 | ADDR2: 000.000.000.000

~ADDR1: 255.255.255.255

subnet 192.168.1.30 192.168.0.0 255.255.0.0

Is 192.168.001.030 in subnet? si

calc -1 f

oop\_exercise\_02: /mnt/c/Users/yurok/Desktop/Учеба/OOP/LABS/02/main.cpp:63: void IPAddress::fromString(std::string): Assertion `isdigit(str[i]) || str[i] == '.'' failed.

Aborted

testcase02

$ ./oop\_exercise\_02

enter 'exit' to quit program, enter 'calc' <ADDR1> <ADDR2>

or 'subnet' <ADDR> <NET> <MASK> to check for subnet

addresses are entered in 123.123.123.123 format

calc 100.50.100.100 100.50.100.100

ADDR1: 100.050.100.100 ADDR2: 100.050.100.100

ADDR1 - ADDR2: 000.000.000.000

ADDR1 + ADDR2: 200.100.200.200

ADDR1 < ADDR2: 0

ADDR1 > ADDR2: 0

ADDR1 = ADDR2: 1

ADDR1 & ADDR2: 100.050.100.100

ADDR1 | ADDR2: 100.050.100.100

~ADDR1: 155.205.155.155

calc 100.50.100.100 100.100.50.100

ADDR1: 100.050.100.100 ADDR2: 100.100.050.100

ADDR1 - ADDR2: 000.206.050.000

ADDR1 + ADDR2: 200.150.150.200

ADDR1 < ADDR2: 1

ADDR1 > ADDR2: 0

ADDR1 = ADDR2: 0

ADDR1 & ADDR2: 100.032.032.100

ADDR1 | ADDR2: 100.032.032.100

~ADDR1: 155.205.155.155

calc 123.123.123.123 123.123.123.123

ADDR1: 123.123.123.123 ADDR2: 123.123.123.123

ADDR1 - ADDR2: 000.000.000.000

ADDR1 + ADDR2: 246.246.246.246

ADDR1 < ADDR2: 0

ADDR1 > ADDR2: 0

ADDR1 = ADDR2: 1

ADDR1 & ADDR2: 123.123.123.123

ADDR1 | ADDR2: 123.123.123.123

~ADDR1: 132.132.132.132

calc 100.100.100.50 100.50.100.100

ADDR1: 100.100.100.050 ADDR2: 100.050.100.100

ADDR1 - ADDR2: 000.050.000.206

ADDR1 + ADDR2: 200.150.200.150

ADDR1 < ADDR2: 0

ADDR1 > ADDR2: 1

ADDR1 = ADDR2: 0

ADDR1 & ADDR2: 100.032.100.032

ADDR1 | ADDR2: 100.032.100.032

~ADDR1: 155.155.155.205

calc 255.255.85.85 255.255.170.170

ADDR1: 255.255.085.085 ADDR2: 255.255.170.170

ADDR1 - ADDR2: 000.000.171.171

ADDR1 + ADDR2: 254.254.255.255

ADDR1 < ADDR2: 1

ADDR1 > ADDR2: 0

ADDR1 = ADDR2: 0

ADDR1 & ADDR2: 255.255.000.000

ADDR1 | ADDR2: 255.255.000.000

~ADDR1: 000.000.170.170

1. Листинг программы

// Шараковский Юрий Дмитриевич. М8О-206Б-19, МАИ. 09.2020.

// Лабораторная работа №2. Условие:

//

// 18 Создать класс IPAddress для работы с адресом в интернет. Класс состоит из четырех чисел unsigned char (a,b,c,d).

// Реализовать арифметические операции сложения, вычитания, а также операции сравнения

// (для сравнение на больше/меньше считать что левые байты главнее т.е. вначале сравниваются первые байты, потом вторые и т.д.).

// Так же реализовать функцию, которая будет определять принадлежность адреса к подсети по адресу подсети (a1,b1,c1,d1)

// и битовой маске подсети (a2,b2,c2,d2). Например, адрес 192.168.1.30 принадлежит подсети 192.168.0.0 с маской 255.255.0.0.

// Операции сложения, вычитания и сравнения необходимо реализовать в виде перегрузки операторов.

// Необходимо реализовать пользовательский литерал для работы с константами типа IPAddress.

#include <iostream>

#include <sstream>

#include <iomanip>

#include <string>

#include <cassert>

class IPAddress {

private:

unsigned char a,b,c,d;

public:

IPAddress(unsigned char a = 0, unsigned char b = 0, unsigned char c = 0, unsigned char d = 0);

void fromString(std::string str);

/\*\*

\* Checks whether an ip address is in a subnet

\* @param addr Net IP address

\* @param mask Net Mask

\* @return 'true' if conditions are met

\*/

bool inSubnet(const IPAddress& netAddr, const IPAddress& netMask) const noexcept;

friend std::istream& operator>>(std::istream& is, IPAddress& addr);

friend std::ostream& operator<<(std::ostream& os, const IPAddress& addr);

friend IPAddress operator ""\_ip(const char\* str, size\_t size);

friend IPAddress operator+(const IPAddress& a, const IPAddress& b);

friend IPAddress operator-(const IPAddress& a, const IPAddress& b);

friend IPAddress operator&(const IPAddress& a, const IPAddress& b);

friend IPAddress operator|(const IPAddress& a, const IPAddress& b);

friend IPAddress operator~(const IPAddress& a);

friend bool operator<(const IPAddress& a, const IPAddress& b);

friend bool operator>(const IPAddress& a, const IPAddress& b);

friend bool operator==(const IPAddress& a, const IPAddress& b);

};

IPAddress::IPAddress(unsigned char a, unsigned char b, unsigned char c, unsigned char d) {

this->a = a;

this->b = b;

this->c = c;

this->d = d;

}

void IPAddress::fromString(std::string str) {

std::string s;

for (size\_t i = 0; i < str.size(); ++i) {

assert(isdigit(str[i]) || str[i] == '.');

if (str[i] == '.')

s.push\_back(' ');

else

s.push\_back(str[i]);

}

std::stringstream ss;

ss << s;

ss >> \*this;

}

std::istream& operator>>(std::istream& is, IPAddress& addr) {

unsigned int a, b, c, d;

is >> a >> b >> c >> d;

addr.a = a;

addr.b = b;

addr.c = c;

addr.d = d;

return is;

}

std::ostream& operator<<(std::ostream& os, const IPAddress& addr) {

os

<< std::setfill('0') << std::setw(3) << (uint32\_t)(addr.a) << '.'

<< std::setfill('0') << std::setw(3) << (uint32\_t)(addr.b) << '.'

<< std::setfill('0') << std::setw(3) << (uint32\_t)(addr.c) << '.'

<< std::setfill('0') << std::setw(3) << (uint32\_t)(addr.d);

return os;

}

IPAddress operator "" \_ip(const char\* str, size\_t size) {

std::string in = str;

IPAddress addr;

addr.fromString(str);

return addr;

}

IPAddress operator+(const IPAddress& a, const IPAddress& b) {

IPAddress res;

res.a = a.a + b.a;

res.b = a.b + b.b;

res.c = a.c + b.c;

res.d = a.d + b.d;

return res;

}

IPAddress operator-(const IPAddress& a, const IPAddress& b) {

IPAddress res;

res.a = a.a - b.a;

res.b = a.b - b.b;

res.c = a.c - b.c;

res.d = a.d - b.d;

return res;

}

IPAddress operator&(const IPAddress& a, const IPAddress& b) {

IPAddress res;

res.a = a.a & b.a;

res.b = a.b & b.b;

res.c = a.c & b.c;

res.d = a.d & b.d;

return res;

}

IPAddress operator|(const IPAddress& a, const IPAddress& b) {

IPAddress res;

res.a = a.a & b.a;

res.b = a.b & b.b;

res.c = a.c & b.c;

res.d = a.d & b.d;

return res;

}

IPAddress operator~(const IPAddress& a) {

IPAddress res;

res.a = ~a.a;

res.b = ~a.b;

res.c = ~a.c;

res.d = ~a.d;

return res;

}

bool operator<(const IPAddress& a, const IPAddress& b) {

bool x = (a.c == b.c ? (a.d < b.d) : false);

bool y = (a.b == b.b ? (a.c < b.c ? true : x) : false);

bool z = (a.a == b.a ? (a.b < b.b ? true : y) : false);

return (a.a < b.a ? true : z);

}

bool operator>(const IPAddress& a, const IPAddress& b) {

bool x = (a.c == b.c ? (a.d > b.d) : false);

bool y = (a.b == b.b ? (a.c > b.c ? true : x) : false);

bool z = (a.a == b.a ? (a.b > b.b ? true : y) : false);

return (a.a > b.a ? true : z);

}

bool operator==(const IPAddress& a, const IPAddress& b) {

return a.a == b.a && a.b == b.b && a.c == b.c && a.d == b.d;

}

bool IPAddress::inSubnet(const IPAddress& addr, const IPAddress& mask) const noexcept {

//example

//addr: 11000000 10101000 00000001 00011110

//net: 11000000 10101000 00000000 00000000

//mask: 11111111 11111111 00000000 00000000

//return true

return ((\*this & mask) == (addr & mask));

}

int main() {

std::string input;

std::cout

<< "enter \'exit\' to quit program, enter \'calc\' <ADDR1> <ADDR2>\n"

<< "\tor \'subnet\' <ADDR> <NET> <MASK> to check for subnet\n"

<< "\taddresses are entered in 123.123.123.123 format"

<< std::endl;

IPAddress mask = "255.255.255.255"\_ip;

IPAddress net = "0.0.0.0"\_ip;

while (true) {

std::cin >> input;

if (input == "exit")

break;

else if (input == "calc")

{

std::string in1, in2;

IPAddress a;

IPAddress b;

std::cin >> in1 >> in2;

a.fromString(in1);

b.fromString(in2);

std::cout << "ADDR1: " << a << "\tADDR2: " << b << std::endl;

std::cout << "ADDR1 - ADDR2: \t" << (a - b) << std::endl;

std::cout << "ADDR1 + ADDR2: \t" << (a + b) << std::endl;

std::cout << "ADDR1 < ADDR2: \t" << (a < b) << std::endl;

std::cout << "ADDR1 > ADDR2: \t" << (a > b) << std::endl;

std::cout << "ADDR1 = ADDR2: \t" << (a == b) << std::endl;

std::cout << "ADDR1 & ADDR2: \t" << (a & b) << std::endl;

std::cout << "ADDR1 | ADDR2: \t" << (a | b) << std::endl;

std::cout << "~ADDR1: \t" << (~a) << std::endl;

}

else if (input == "subnet")

{

std::string in1, in2, in3;

IPAddress a;

std::cin >> in1 >> in2 >> in3;

a.fromString(in1);

net.fromString(in2);

mask.fromString(in3);

std::cout << "Is " << a << " in subnet? ";

if (a.inSubnet(net, mask))

std::cout << "si";

else

std::cout << "ni";

std::cout << std::endl;

}

}

}

1. Выводы

Язык C++ имеет гибкие механизмы использования стандартных операторов для работы с определенными пользователем классами, которые можно использовать в том числе и для упрощения программирования, например, можно писать A + B, вместо A.Add(B) для каких либо объектов A и B пользовательского класса. Пользовательские литералы также упрощают определение констант класса в коде с помощью чисел или строк, наиболее очевидное применение - литералы для класса Distance обозначающие расстояние в сантиметрах, метрах и километрах (cm, m, km) соответственно.

Список литературы

1. C++ Primer (5th Edition) Stanley B. Lippman, Josee Lajoie, E. Moo.

2. Effective Modern C++, Scott Meyers.