Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра вычислительных систем

Курсовая работа на тему:

**IPCalc**

Выполнил: студент 4 курса,

гр. ЗП-91

Починок С.В.

Проверил: Токмашева Е.И.

Новосибирск 2022

Содержание

Описание работы 3

Описание файлов проекта 4

Алгоритмы работы 6

Скриншоты 7

Листинг кода 9

**Описание работы.**

Тип приложения: консольное приложение

Среда разработки: QT Creator v. 4.5.1 (QT 5.10.1)

Язык разработки: C++

Git: https://github.com/Kvadrian/IPcalcTRPO

Принцип работы:

Пользователь запускает консольное приложение. Далее ему предлагается ввести IP-адрес (Host) по одному значению каждого из 4х октетов по порядку (допустимые значения 0-255). Т.е. В самом начале вводится первый октет, потом второй и т.д.

Далее пользователь вводит маску подсети (допустимые значения от 1 до 32). После этого программа должна посчитать подстановочную маску (Wildcard), подсеть (Network), Широковещательный адрес (Broadcast), первый и последний адреса в подсети (HostMin, HostMax, с-но). Представление адресов должно быть, как в десятичном, так и в двоичном формате.

**Описание файлов проекта**

**ip\_calc.h** – заголовочный файл с объявлением функций и библиотеками vector, string, sstring для работы со строками и неопределенными массивами char

**ip\_calc.сpp** – в файле представлены основные функции проекта

**get\_IP\_as\_string\_bin –** запись IP адреса в двоичном виде в строку output

**get\_IP\_as\_string\_dec –** запись IP адреса в десятичном виде в строку output

**ip\_calc::calculate() –** основной метод подсчета искомых значений с вызовов основных функций подсчета

**calculate\_broadcast** – функция нахождения широковещательного адреса путем побитовой дизъюнкиции адреса хоста и подстановочной маски

**calculate\_hostmax** - функция нахождения максимального адреса подсети путем побитовой дизъюнкции последнего двоичного представления широковещательного адреса и 1

**calculate\_hostmin** – функция нахождения минимального адреса подсети путем побитовой дизъюнкции последнего двоичного представления адреса подсети и 1

**calculate\_network** – функция нахождения адреса подсети путем побитовой конъюкции IP адреса (Host) и сетевой маски

**calculate\_wildcard** – функция нахождения подстановочной маски, путем инверсии побитового представления сетевой маски

**reset\_address** – метод для сброса (заполение нулями), нужного октета в двоичном представлении адреса

**set\_address** – перевод в двоичное представление, в нужном октете, десятичного представления

**bool set\_address** – проверка на корректность ввода значения (по умолчанию от 0 до 255), через вызов функции validate и запись адреса в десятичном представлении

**set\_netmask** – перевод в двоичное представление десятичного значения сетевой маски

**bool set\_netmask** - проверка на корректность ввода значения (по умолчанию от 1 до 32), через вызов функции validate и запись адреса в десятичном представлении

**validate** – функция проверки на корректность ввода пользователем числа

**unit\_tests.cpp**

**mask\_test –** массивстрок с корректными (числовые положительные значения ) и некорректными значениями, которые может ввести пользователь (буквенные значения, отрицательные числа, знаки переноса)

**TRPO.cpp**

Основной файл, в котором ведется работа

**write\_line –** метод вывода значения адреса в десятичном виде, затем, через табуляцию, двоичного представления, через вызов функций get\_IP\_as\_string\_dec и get\_IP\_as\_string\_bin

**Алгоритм работы основного файла**

Вводим строку buffer для записи адреса

Выводим на экран сообщение, с просьбой ввести пользователю IP адрес по октетам (для этого вводится цикл for, значения записываются в buffer), в случае некорректного ввода, предлагается ввести значение заново. Итерация цикла при этом не увеличивается.

Выводим на экран сообщение, с просьбой ввести пользователю маску (для этого вводится цикл while, значения записываются в buffer). В случае некорректного ввода, выход из цикла не осуществляется

Далее выводятся значения IP адреса и маски подсети, в десятичном и двоичном виде, через вызов метода write\_line c проставлением значений функций get\_address и get\_netmask

Создаем неопределенный массив result, знаечния в который записываем через вызов функции calculate

Далее вывод оставшихся значений через вызов write\_line

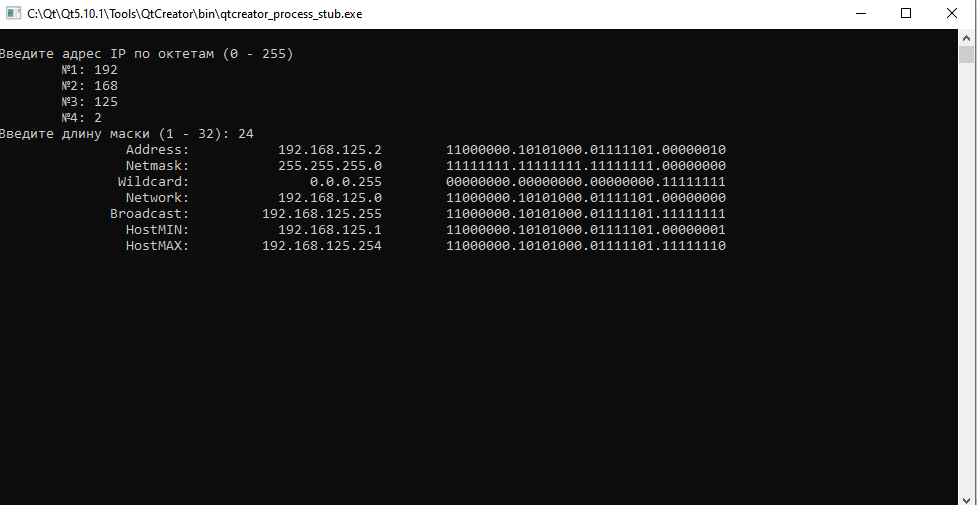
Алгоритм работы unit\_tests.cpp

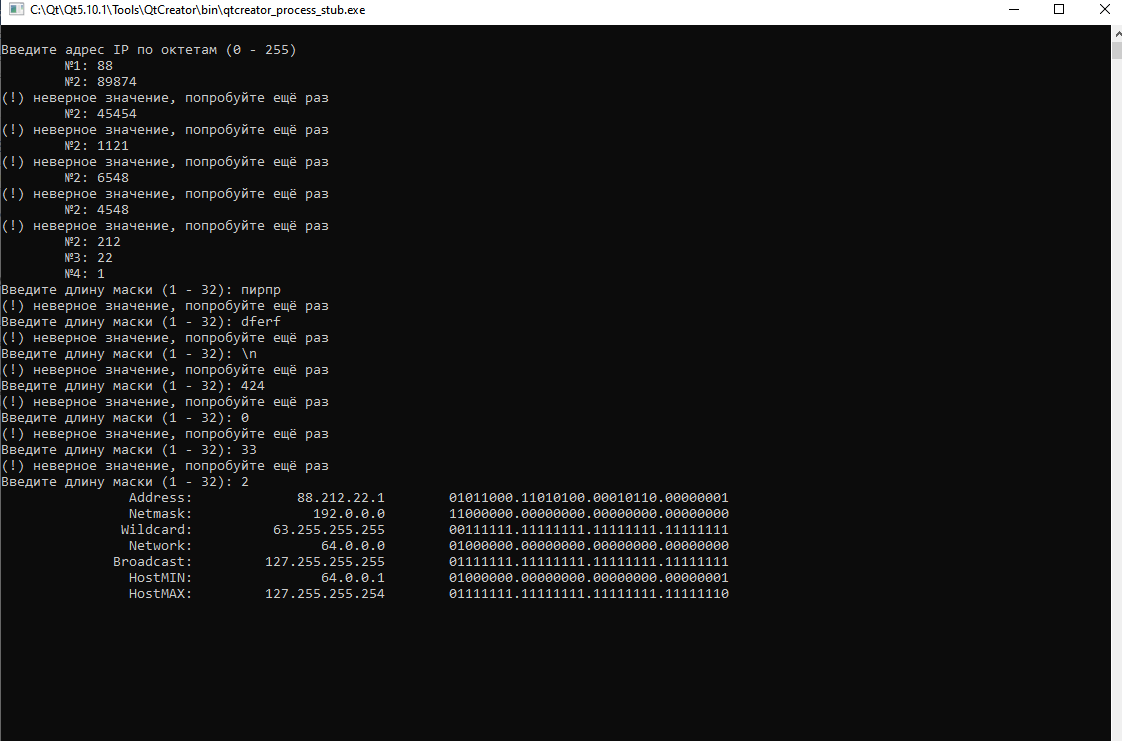
Корректность работы основного файла зависит от данных, введенных, пользователем, поэтому нам необходимо проверить работу только двух функций validate и set\_netmask – с-но. Для этого проверяем работоспособность обеих функций, через значения, заданные в mask\_test.

Проверка была на значениях "1", "0","-1","32","/n","33","-99999999999999999999999999999999999999999999","9999999999999999999999999999999999999999999999999999999999999999999999999999","\t","256","mmmmm"

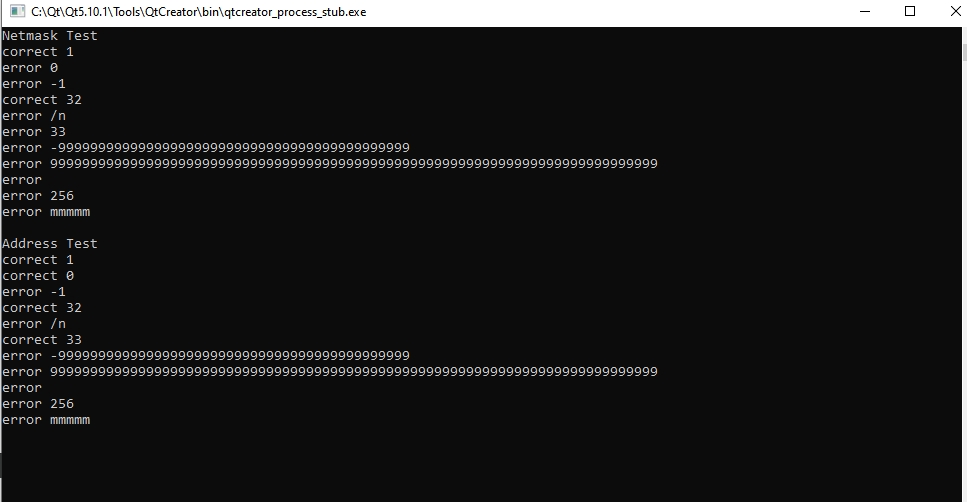
**Скриншоты работы**

Основной файл:





**Тесты:**



**Листинг кода**

ip\_calc.h

#ifndef IP\_CALC\_H

#define IP\_CALC\_H

#include"string"

#include"sstream"

#include"vector"

const std::string **get\_IP\_as\_string\_bin**(const unsigned int&IP);

const std::string **get\_IP\_as\_string\_dec**(const unsigned int&IP);

class ip\_calc{

public:

ip\_calc();

ip\_calc(const unsigned int&address,const unsigned int&netmask);

/\* \* \* INTERFACE \* \* \*/

std::vector<unsigned int>**calculate**();

const unsigned int&**get\_address**();

const unsigned int&**get\_netmask**();

void **set\_address**(const unsigned char&octet\_from\_0\_to\_255,const unsigned char&pos\_from\_1\_to\_4\_where\_1st\_is\_the\_most\_significant);

bool **set\_address**(const std::string&octet\_from\_0\_to\_255,const unsigned char&pos\_from\_1\_to\_4\_where\_1st\_is\_the\_most\_significant);

void **set\_netmask**(const unsigned char&length\_from\_1\_to\_32);

bool **set\_netmask**(const std::string&length\_from\_1\_to\_32);

bool **validate**(const std::string&number,const int&from,const int&to);

private:

/\* \* \* FIELDS \* \* \*/

struct{

unsigned int address;

unsigned int netmask;

} IP;

/\* \* \* METHODS \* \* \*/

unsigned int **calculate\_broadcast**(const unsigned int&network,const unsigned int&wildcard);

unsigned int **calculate\_hostmax**(const unsigned int&wildcard,const unsigned int&broadcast);

unsigned int **calculate\_hostmin**(const unsigned int&network);

unsigned int **calculate\_network**();

unsigned int **calculate\_wildcard**();

void **reset\_address**(const unsigned char&pos);

void **reset\_netmask**();

};

#endif // IP\_CALC\_H

ip\_calc.cpp

#include"ip\_calc.h"

const std::string **get\_IP\_as\_string\_bin**(const unsigned int&IP){

std::stringstream output;

for(unsigned char i=32;i!=0;){

output<<(1&(IP>>--i));

if(i!=0&&i%8==0)output<<'.';

}

return output.str();

}

const std::string **get\_IP\_as\_string\_dec**(const unsigned int&IP){

std::stringstream output;

for(unsigned char i=4;i!=0;){

output<<(0xff&(IP>>(8\*--i)));

if(i!=0)output<<'.';

}

return output.str();

}

ip\_calc::**ip\_calc**(){IP={0,0};}

ip\_calc::**ip\_calc**(const unsigned int&address,const unsigned int&netmask){IP={address,netmask};}

std::vector<unsigned int>ip\_calc::**calculate**(){

std::vector<unsigned int>result;

result.push\_back(calculate\_wildcard());

result.push\_back(calculate\_network());

result.push\_back(calculate\_broadcast(result.at(0),result.at(1)));

result.push\_back(calculate\_hostmin(result.at(1)));

result.push\_back(calculate\_hostmax(result.at(0),result.at(2)));

return result;

}

unsigned int ip\_calc::**calculate\_broadcast**(const unsigned int&network,const unsigned int&wildcard){return network|wildcard;}

unsigned int ip\_calc::**calculate\_hostmax**(const unsigned int&wildcard,const unsigned int&broadcast){

return (wildcard&1)==(broadcast&1)?

broadcast&(~static\_cast<unsigned int>(1)):

broadcast|1;

}

unsigned int ip\_calc::**calculate\_hostmin**(const unsigned int&network){return network|1;}

unsigned int ip\_calc::**calculate\_network**(){return IP.address&IP.netmask;}

unsigned int ip\_calc::**calculate\_wildcard**(){return ~IP.netmask;}

const unsigned int&ip\_calc::**get\_address**(){return IP.address;}

const unsigned int&ip\_calc::**get\_netmask**(){return IP.netmask;}

void ip\_calc::**reset\_address**(const unsigned char&pos){

if(pos>0&&pos<=4)IP.address&=~((static\_cast<unsigned int>(0xff))<<(8\*(4-pos)));

}

void ip\_calc::**reset\_netmask**(){IP.netmask=0;}

void ip\_calc::**set\_address**(const unsigned char&octet,const unsigned char&pos){

if(pos>0&&pos<=4){

reset\_address(pos);

IP.address|=((static\_cast<unsigned int>(octet))<<(8\*(4-pos)));

}}

bool ip\_calc::**set\_address**(const std::string&octet,const unsigned char&pos){

if(pos>0&&pos<=4){

if(!validate(octet,0,255))return false;

else{

set\_address(static\_cast<unsigned char>(std::stoi(octet)),pos);

return true;

}}

else return false;

}

void ip\_calc::**set\_netmask**(const unsigned char&length){

if(length>0&&length<=32){

reset\_netmask();

for(unsigned char i=length;i!=0;i--)

IP.netmask|=static\_cast<unsigned int>(1)<<(32-i);

}}

bool ip\_calc::**set\_netmask**(const std::string&length){

if(!validate(length,1,32))return false;

else{

set\_netmask(static\_cast<unsigned char>(std::stoi(length)));

return true;

}}

bool ip\_calc::**validate**(const std::string&number,const int&from,const int&till){

for(auto i=number.size();i-->0;)

if(number.at(i)<'0'||number.at(i)>'9')return false;

try{

int \_number=std::stoi(number);

return \_number>=from&&\_number<=till;

}

catch(...){

return false;

}}

TRPO.cpp

#include"iomanip"

#include"iostream"

#include"ip\_calc.h"

using namespace std;

void **write\_line**(const char\*header,const unsigned int&value){

cout<<setw(24)<<header

<<setw(24)<<get\_IP\_as\_string\_dec(value)

<<'\t'<<get\_IP\_as\_string\_bin(value)

<<endl;

}

int **main**(){

setlocale(LC\_ALL,"rus");

ip\_calc Host\_adr;

// ВВОД

string buffer;

cout<<endl

<<"Введите адрес IP по октетам (0 - 255)"<<endl;

for(unsigned short i=1;i<=4;){

cout<<"\t№"<<i<<": ";

cin>>buffer;

if(Host\_adr.set\_address(buffer,i))i++;

else cout<<"(!) неверное значение, попробуйте ещё раз"<<endl;

}

while(true){

cout<<"Введите длину маски (1 - 32): ";

cin>>buffer;

if(Host\_adr.set\_netmask(buffer))break;

else cout<<"(!) неверное значение, попробуйте ещё раз"<<endl;

}

// ВЫВОД

write\_line("Address:",Host\_adr.get\_address());

write\_line("Netmask:",Host\_adr.get\_netmask());

vector<unsigned int>result=Host\_adr.calculate();

const char\*header[]={"Wildcard:","Network:","Broadcast:","HostMIN:","HostMAX:"};

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

write\_line(header[i],result.at(i));

return EXIT\_SUCCESS;

unit\_tests.cpp

#include"ip\_calc.h"

#include"iostream"

#include"iomanip"

using namespace std;

int **main**()

{

int n=11;

ip\_calc UT;

std::string mask\_test [n]= {

"1",

"0",

"-1",

"32",

"/n",

"33",

"-99999999999999999999999999999999999999999999",

"9999999999999999999999999999999999999999999999999999999999999999999999999999",

"\t",

"256",

"mmmmm"

};

cout<<"Netmask Test";

cout<<"\n";

for(int i=0;i<n;i++)

{

if ((UT.set\_netmask(mask\_test[i]))==false)

cout<<"error "<<mask\_test[i]<<"\n";

else

cout<<"correct "<<mask\_test[i]<<"\n";

}

cout<<"\n";

cout<<"Address Test \n";

for(int i=0;i<n;i++)

{

if (UT.validate(mask\_test[i],0,255))

cout<<"correct "<<mask\_test[i]<<"\n";

else

cout<<"error "<<mask\_test[i]<<"\n";

}

return EXIT\_SUCCESS;

}