Написал свой IP-калькулятор на языке программирования Си, который можно доработать при необходимости. Произвел рефакторинг кода.

В начале объявляем заголовочные файлы. В данной программе используются стандартные заголовочные файлы, установка дополнительных пакетов не требуется.

#include <stdio.h> // printf()

#include <stdlib.h> // strtol() - converting string to int

#include <string.h> // strlen() - length string

#include <errno.h> // For errno - fast exit from function, if appearance error

#include <math.h> // for pow(), added for Win, not added for Linux

//#include <ctype.h> // islower() | older function usage in this code

#include <arpa/inet.h> // inet\_pton(), fast check argv[1] on conformity syntax IPv4

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Объявляем переменные. Часть переменных пришлось вынести в начало программы, т. к. функции должны быть объявлены, чтобы их можно было повторно использовать в других функциях.

// define envs:

char mask; // argv[2]

int oct\_1 = 0, oct\_2 = 0, oct\_3 = 0, oct\_4 = 0; // env for octet in argv1 (just print argument), split IPv4 on 4 octets

int int\_mask = 0, active\_oct = 0; // int mask, select active octet by mask

int ip\_gate = 0, ip\_min = 0, ip\_max = 0, l = 0, ip\_broadcast = 0; // dynamic envs for print octet result

int count = 1; // calculate count potencial hosts in network by mask

const int zero = 0, const\_gate = 1, const\_broadcast = 255; // envs constant for print octet result

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Объявление функций должно происходить вначале программы, чтобы затем функции можно было вызывать в основной функции main. Не смотрел, как можно объявлять функции после основной функции main для улучшения читаемости кода.

Функции calc\_min\_max, calc\_0, calc\_1\_7, calc\_8\_15, calc\_16\_23, calc\_24\_30, call\_switch, производят основную часть вычислений.

int calc\_min\_max, int calc\_0, int calc\_1\_7, int calc\_8\_15, int calc\_16\_23, int calc\_24\_30, int call\_switch

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Вначале происходит проверка на наличие аргументов запущенной программы. Если аргументов нет – выводится сообщение и программа закрывается. Первый аргумент argc = 1, это **argv[0]**, т. е. имя запущенной программы. Если эта часть кода отсутствует – выводится ошибка **"segmentation fault"**. Эта и последующие ошибки нумеруются (самым простым способом – вручную).

if(argc < 2) { printf("Error #0000. No arguments. Please enter IP-address and mask (example 192.168.1.1 24).\n");

exit(1); }

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Объявляем массив **\*str\_ipv4**, который будет содержать либо IP-адрес, либо «--help». указанный в первом аргументе. Объявляем аргумент **\*chk\_arg\_help**, который будет использоваться для проверки, что мы вызвали справку. Затем производится сравнение, что первый аргумент равен «--help». В текущем коде достаточно любого вхождения, в т. ч. первого текстового литерала «-». Выводится сообщение и программа закрывается.

При сборке и установке бинарного пакета будет доступна справочная страница («**man ip-range**»). Ролик со сборкой будет позже.

if ( \*str\_ipv4 == \*chk\_arg\_help )

{ // hardcode

printf("usage man ip-range\n");

printf("example: ip-range 2.193.104.207 16\n");

printf("------------------------------------------------------\n");

printf("IP-address:\t\t\t\t[2.193.104.207]\n");

printf("IP-mask:\t\t\t\t[16]\n");

printf("Network:\t\t\t\t[2.193.0.0/16]\n");

printf("Count IP-address:\t\t\t[65534]\n");

printf("IP-address default gateway:\t\t[2.193.0.1]\n");

printf("IP-address min:\t\t\t\t[2.193.0.2]\n");

printf("IP-address max:\t\t\t\t[2.193.255.254]\n");

printf("IP-address broadcast:\t\t\t[2.193.255.255]\n");

printf("------------------------------------------------------\n");

exit(1);

}

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Если мы указали аргумент, но не «--help», значит это должно быть 2 аргумента: IP-адрес и маска (через пробел). Если мы указали один аргумент, то выводится сообщение и программа закрывается. Я не стал реализовывать проверку, что аргументов равно 3 или меньше четырех. Если аргументов больше, то четвертый и последующие аргументы обрабатываться не будут.

if(argc < 3) { printf("Error #0001. Please enter IP-address and mask (example 192.168.1.1 24).\n");

exit(1); }

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Производим проверку, что первый аргумент является IP-адресом (т. е. 4 числа от 0 до 255 с разделителем точкой). Ручная реализация кода требовала много времени и много кода, я воспользовался стандартным заголовочным файлом **<arpa/inet.h>**, который реализует эту проверку. Если первый аргумент не является IPv4-адресом, то выводится сообщение и программа закрывается.

int n\_1 = strlen(str\_ipv4);

int ip\_version(const char \*src) {

char buf[INET6\_ADDRSTRLEN];

if (inet\_pton(AF\_INET, src, buf)) {

return 4; }

return -1; }

if ( ip\_version(str\_ipv4) != 4 ) {

printf("Error #0002. 1st argument not a IPv4 address (example: %s 192.168.1.0 24).\n", argv[0]);

exit(1);

}

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Производим проверку валидности маски, т. е. что второй аргумент является числом от 0 до 32. Объявляем массив **\*p** (здесь код не оптимален, не стал тратить время на оптимизацию), переменная **mask** получает значение элемента массива (в Си это важно уметь делать). Если строковый массив состоит из цифр (целочисленное значение), то его можно объявить числом, объявив числовую переменную. Далее в программе используется маска сети из переменной **int\_mask**. Если маска является числом от 0 до 32, то используется оператор безусловного перехода **gotoprogramm\_2:**, если нет, выводится сообщение и программа закрывается. Обычно, оператор Goto рекомендуют не использовать.

char \*p;

mask = strtol(argv[2], &p, 10);

int\_mask = mask;

switch(int\_mask) // use check for 'case' - simple and work condition '0 =< x >= 32'

{

case 0 ... 32:

goto gotoprogramm\_2; //go run main programm, mask is correctly

return 0;

default:

printf("Error #0003. 2nd argument != 0-32.\n"); // mask is not correctly, show message and exit programm

exit(1);

return 0;

}

gotoprogramm\_2:

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Если предыдущие проверки прошли, значит у нас указаны корректные аргументы. Объявляем массив **\*ip\_char\_oct**, который через разделитель «.» делится на 4 числовых октета. Функция **atoi** позволяет конвертировать входную текстовую строку в число до первого тестового литерала, не являющегося цифрой. Т. к. мы уже знаем, что IPv4-адрес валидный, то нам может встретиться только «.», поэтому алгоритм конвертации текста в числа простой и не требует доработок. На выходе получаем 4 переменные = 4 октета.

char \*ip\_char\_oct = argv[1];

char \*char\_oct = strtok (ip\_char\_oct, ".");

for (int i = 1; i < 5; i++)

{

if (i == 1) oct\_1 = atoi(char\_oct); // int oct\_1 value

if (i == 2) oct\_2 = atoi(char\_oct); // int oct\_2 value

if (i == 3) oct\_3 = atoi(char\_oct); // int oct\_3 value

if (i == 4) oct\_4 = atoi(char\_oct); // int oct\_4 value

char\_oct = strtok(NULL, ".");

}

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Т. к. на текущий момент в программе мы получили IP-адрес и маску в числовом виде, то выводим их в консоль. Это абсолютно не то же самое, если вывести в консоль argv[1] и argv[2].

printf("IP-address:\t\t\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, oct\_3, oct\_4);

printf("IP-mask:\t\t\t\t[%d]\n", int\_mask); // show masks

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Вычисляем степень, и считаем количество устройств в подсети **int\_mask**. Для подсетей /0, /31, /32 переменная **х** не используется.

int x = 32 - int\_mask;

// calculate count hosts by mask

int i = x;

while (i > 0) {

count = count \* 2; // calculate count potencial hosts in network by mask

i--; }

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Производится сравнение **int\_mask** со значением от 1 до 30, если условие выполняется, то вызывается функция **calc\_min\_max** с аргументом **int\_mask**. Функция **calc\_min\_max** на основе маски подсети **int\_mask** вычисляет какой октет является активным. Активный октет будет использоваться при вычислении минимального и максимального IP-адреса в указанной подсети. После вычисления производится переход к метке **result:**. Была попытка указать динамическое значение в условие switch-case, в зависимости от маски можно было динамически вычислять активный октет и значение текущего октета (какая /27 в /24 маске), но Си так не умеет (доступно в С++). Это было бы важно если бы был большой разброс значений 1-1.000.000 и не было бы привязки к маске (степень 2). Тогда кода было бы слишком много и вычисление занимало много времени (относительно решаемой задачи).

switch(int\_mask) {

// can calculate with usage degree

case 1 ... 30:

calc\_min\_max(int\_mask);

goto result;

return 0; }

int calc\_min\_max(int min\_max\_mask) {

switch(min\_max\_mask) // use check for 'case' - simple and work condition '1 =< x >= 30'

{

case 1 ... 7: // exclude /0, because whole network

active\_oct = oct\_1;

return active\_oct;

case 8 ... 15:

active\_oct = oct\_2;

return active\_oct;

case 16 ... 23:

active\_oct = oct\_3;

return active\_oct;

case 24 ... 30: // exclude /31, /32, because network without free pool IP-address

active\_oct = oct\_4;

return active\_oct; }

}

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Вызывается функция **call\_switch**, которая в зависимости от маски подсети **int\_mask** вызывает функцию, которая будет вычислять IP-адреса: шлюз сети по умолчанию, минимальный IP-адрес, максимальный IP-адрес, широковещательный IP-адрес. Если маска равна 0, то можно было сразу вывести текст (hardcode), без вызова еще одной функции  **calc\_0**. В данном случае я придерживался унификации кода (в данном случае код выполняется быстро).

call\_switch(active\_oct, int\_mask); // function place in outside file "ip-range-min-max"

int call\_switch(int active\_oct, int int\_mask) {

switch(int\_mask) // execute function for calculate min-max IP-address

{

case 0:

calc\_0 (int\_mask);

case 1 ... 7:

calc\_1\_7(int\_mask);

case 8 ... 15:

calc\_8\_15(int\_mask);

case 16 ... 23:

calc\_16\_23(int\_mask);

case 24 ... 30:

calc\_24\_30(int\_mask);

return 0;

case 31:

printf("2 IP-address in network: gateway (even) and broadcast (not even).\n"); // hardcode

exit(1);

return 0;

case 32:

printf("1 IP-address is whole network.\n"); // hardcode

exit(1);

return 0;

// default:

// return 0; }

}

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

В зависимости от маски подсети вычислен активный октет, в активном октете производится вычисление. Шлюз по умолчанию тоже является доступным IP-адресом, но т. к. достаточно часто он указан на сетевом устройстве, а так же я встречал в iLO зашитое в прошивку вычисление шлюза по умолчанию (иначе не сохраняет настройки), то шлюз в данном случае считается отдельно. Это поведение можно изменить в коде и перекомпилировать. Для подсети /0 выводится hardcode. Я разделил вычисление на первое вхождение в цикле и последующие вхождения, иначе получался сложный алгоритм.

int calc\_0(int int\_mask) {

printf("Network:\t\t\t\t[%d.%d.%d.%d/%d]\n", zero, zero, zero, zero, int\_mask);

printf("Count IP-address (max in IPv4):\t\t[4294967294]\n"); // (2^32) - 2 = 4294967294

// default gateway

printf("IP-address default gateway (unlikely):\t[%d.%d.%d.%d]\n", zero, zero, zero, const\_gate);

// min-max address

printf("IP-address min:\t\t\t\t[%d.%d.%d.2]\n", zero, zero, zero);

printf("IP-address max:\t\t\t\t[%d.%d.%d.254]\n", const\_broadcast, const\_broadcast, const\_broadcast);

printf("IP-address broadcast:\t\t\t[%d.%d.%d.%d]\n", const\_broadcast, const\_broadcast, const\_broadcast, const\_broadcast);

exit(1);

return 0;

}

int calc\_1\_7(int int\_mask)

{

int degree\_1\_7 = 8-int\_mask; // 2^degree\_1\_7 count degree is /8 network

int b = 1; // b = count is /8 network (calculate include 0), if mask 6, b = 4

while (degree\_1\_7 > 0) {

b = b \* 2; // count classing networks /8

degree\_1\_7--;

}

int count\_mask\_1\_7 = b-1; // if mask 6, b = 4, count\_mask\_1\_7 = 3 (first eq 0-3)

if (count\_mask\_1\_7 >= active\_oct) // if first eq

{

ip\_gate = 0; // if first eq

ip\_min = 0; // if first eq

ip\_max = count\_mask\_1\_7;

ip\_broadcast = ip\_max;

// network

printf("Network:\t\t\t\t[%d.%d.%d.%d/%d]\n", ip\_gate, zero, zero, zero, int\_mask);

printf("Count IP-address:\t\t\t[%d]\n", count-2);

// default gateway

printf("IP-address default gateway:\t\t[%d.%d.%d.%d]\n", ip\_gate, zero, zero, const\_gate);

// min-max address

printf("IP-address min:\t\t\t\t[%d.%d.%d.2]\n", ip\_min, zero, zero);

printf("IP-address max:\t\t\t\t[%d.%d.%d.254]\n", ip\_max, const\_broadcast, const\_broadcast);

printf("IP-address broadcast:\t\t\t[%d.%d.%d.%d]\n", ip\_broadcast, const\_broadcast, const\_broadcast, const\_broadcast);

exit(1);

return 0;

}

for (l = b; l <= active\_oct+b; l++) {

ip\_gate = l - b;

ip\_min = ip\_gate;

ip\_max = l-1;

ip\_broadcast = ip\_max;

l = l + (b - 1); // l++ delay

}

if (l >= active\_oct)

{

// network

printf("Network:\t\t\t\t[%d.%d.%d.%d/%d]\n", ip\_gate, zero, zero, zero, int\_mask);

printf("Count IP-address:\t\t\t[%d]\n", count-2);

// default gateway

printf("IP-address default gateway:\t\t[%d.%d.%d.%d]\n", ip\_gate, zero, zero, const\_gate);

// min-max address

printf("IP-address min:\t\t\t\t[%d.%d.%d.2]\n", ip\_min, zero, zero);

printf("IP-address max:\t\t\t\t[%d.%d.%d.254]\n", ip\_max, const\_broadcast, const\_broadcast);

printf("IP-address broadcast:\t\t\t[%d.%d.%d.%d]\n", ip\_broadcast, const\_broadcast, const\_broadcast, const\_broadcast);

exit(1);

return 0; }

return 0;

}

int calc\_8\_15(int int\_mask)

{

int degree\_8\_15 = 16-int\_mask; // 2^degree\_8\_15 count degree is /16 network

int b = 1; // b = count is /16 network (calculate include 0), if mask 14, b = 4

while (degree\_8\_15 > 0) {

b = b \* 2; // count classing networks /16

degree\_8\_15--;

}

int count\_mask\_8\_15 = b-1; // if mask 14, b = 4, count\_mask\_8\_15 = 3 (first eq 0-3)

if (count\_mask\_8\_15 >= active\_oct) // if first eq

{

ip\_gate = 0; // if first eq

ip\_min = 0; // if first eq

ip\_max = count\_mask\_8\_15; // if first eq

ip\_broadcast = ip\_max;

// network

printf("Network:\t\t\t\t[%d.%d.%d.%d/%d]\n", oct\_1, ip\_gate, zero, zero, int\_mask);

printf("Count IP-address:\t\t\t[%d]\n", count-2);

// default gateway

printf("IP-address default gateway:\t\t[%d.%d.%d.%d]\n", oct\_1, ip\_gate, zero, const\_gate);

// min-max address

printf("IP-address min:\t\t\t\t[%d.%d.%d.2]\n", oct\_1, ip\_min, zero);

printf("IP-address max:\t\t\t\t[%d.%d.%d.254]\n", oct\_1, ip\_max, const\_broadcast);

printf("IP-address broadcast:\t\t\t[%d.%d.%d.%d]\n", oct\_1, ip\_broadcast, const\_broadcast, const\_broadcast);

exit(1);

return 0;

}

for (l = b; l <= active\_oct+b; l++) {

ip\_gate = l - b;

ip\_min = ip\_gate;

ip\_max = l-1;

ip\_broadcast = ip\_max;

l = l + (b - 1); // l++ delay

}

if (l >= active\_oct)

{

// network

printf("Network:\t\t\t\t[%d.%d.%d.%d/%d]\n", oct\_1, ip\_gate, zero, zero, int\_mask);

printf("Count IP-address:\t\t\t[%d]\n", count-2);

// default gateway

printf("IP-address default gateway:\t\t[%d.%d.%d.%d]\n", oct\_1, ip\_gate, zero, const\_gate);

// min-max address

printf("IP-address min:\t\t\t\t[%d.%d.%d.2]\n", oct\_1, ip\_min, zero);

printf("IP-address max:\t\t\t\t[%d.%d.%d.254]\n", oct\_1, ip\_max, const\_broadcast);

printf("IP-address broadcast:\t\t\t[%d.%d.%d.%d]\n", oct\_1, ip\_broadcast, const\_broadcast, const\_broadcast);

exit(1);

return 0; }

return 0;

}

int calc\_16\_23(int int\_mask)

{

int degree\_16\_23 = 24-int\_mask; // 2^degree\_16\_23 count degree is /24 network

int b = 1; // b = count is /24 network (calculate include 0), if mask 22, b = 4

while (degree\_16\_23 > 0) {

b = b \* 2; // count classing networks /24

degree\_16\_23--;

}

int count\_mask\_16\_23 = b-1; // if mask 22, b = 4, count\_mask\_16\_23 = 3 (first eq 0-3)

if (count\_mask\_16\_23 >= active\_oct) // if first eq

{

ip\_gate = 0; // if first eq

ip\_min = 0; // if first eq

ip\_max = count\_mask\_16\_23;

ip\_broadcast = ip\_max;

// network

printf("Network:\t\t\t\t[%d.%d.%d.%d/%d]\n", oct\_1, oct\_2, ip\_gate, zero, int\_mask);

printf("Count IP-address:\t\t\t[%d]\n", count-2);

// default gateway

printf("IP-address default gateway:\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, ip\_gate, const\_gate);

// min-max address

printf("IP-address min:\t\t\t\t[%d.%d.%d.2]\n", oct\_1, oct\_2, ip\_min);

printf("IP-address max:\t\t\t\t[%d.%d.%d.254]\n", oct\_1, oct\_2, ip\_max);

printf("IP-address broadcast:\t\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, ip\_broadcast, const\_broadcast);

exit(1);

return 0;

}

for (l = b; l <= active\_oct+b; l++)

{ ip\_gate = l - b;

ip\_min = ip\_gate;

ip\_max = l-1;

ip\_broadcast = ip\_max;

l = l + (b - 1); // l++ delay

}

if (l >= active\_oct)

{

// network

printf("Network:\t\t\t\t[%d.%d.%d.%d/%d]\n", oct\_1, oct\_2, ip\_gate, zero, int\_mask);

printf("Count IP-address:\t\t\t[%d]\n", count-2);

// default gateway

printf("IP-address default gateway:\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, ip\_gate, const\_gate);

// min-max address

printf("IP-address min:\t\t\t\t[%d.%d.%d.2]\n", oct\_1, oct\_2, ip\_min);

printf("IP-address max:\t\t\t\t[%d.%d.%d.254]\n", oct\_1, oct\_2, ip\_max);

printf("IP-address broadcast:\t\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, ip\_broadcast, const\_broadcast);

exit(1);

return 0; }

return 0;

}

int calc\_24\_30(int int\_mask)

{

if (0+count-1 >= active\_oct) // if first eq

{ ip\_gate = l + 1; // if first eq

ip\_min = l + 2; // if first eq

ip\_max = l + (count - 2); // if first eq

ip\_broadcast = l + (count - 1);

// network

printf("Network:\t\t\t\t[%d.%d.%d.%d/%d]\n", oct\_1, oct\_2, oct\_3, ip\_gate-1, int\_mask);

printf("Count IP-address:\t\t\t[%d]\n", count-2);

// default gateway

printf("IP-address default gateway:\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, oct\_3, ip\_gate);

// min-max address

printf("IP-address min:\t\t\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, oct\_3, ip\_min);

printf("IP-address max:\t\t\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, oct\_3, ip\_max);

printf("IP-address broadcast:\t\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, oct\_3, ip\_broadcast);

exit(1);

return 0;

}

for (l = count; l <= active\_oct+count; l++)

{ ip\_gate = l - (count - 1);

ip\_min = l - (count - 2);

ip\_max = l - 2;

ip\_broadcast = l - 1;

l = l + (count - 1); // l++ delay

}

if (l >= active\_oct) {

// network

printf("Network:\t\t\t\t[%d.%d.%d.%d/%d]\n", oct\_1, oct\_2, oct\_3, ip\_gate-1, int\_mask);

printf("Count IP-address:\t\t\t[%d]\n", count-2);

// default gateway

printf("IP-address default gateway:\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, oct\_3, ip\_gate);

// min-max address

printf("IP-address min:\t\t\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, oct\_3, ip\_min);

printf("IP-address max:\t\t\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, oct\_3, ip\_max);

printf("IP-address broadcast:\t\t\t[%d.%d.%d.%d]\n", oct\_1, oct\_2, oct\_3, ip\_broadcast);

exit(1);

return 0; }

return 0;

}