INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Redes de Computadoras

"IP CALCULATOR"

Abstract

An IP CALCULATOR is a program which help the user to calculate the Class, Network Ip, the Hosts Range, the Netmask and the Broadcast Ip only with the Ip introduced.

By:

Eduardo Alberto Pereda Guzmán

Professor:
MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

September 2018

Index

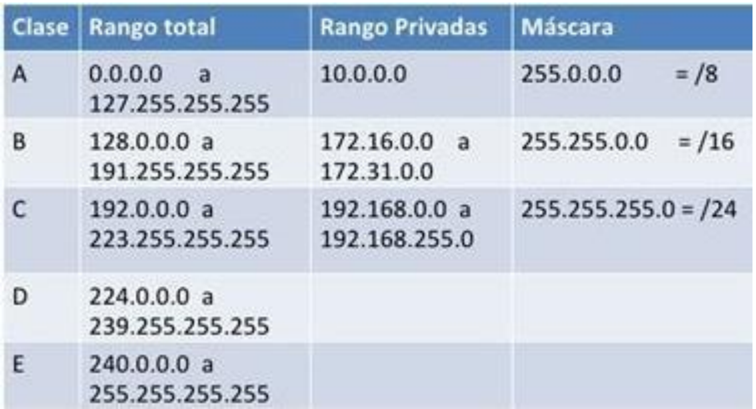
Contenido

Introduction:

First of all, a lot of devices in the world are connected within a global network, which plenty of information is sent daily. Every device connected in this global network has an address, better known as an IP. In addition, this devices also have a MAC address, this components are necessary to transport the information.

Literature review:

Ip classes



Clase	Rango total	Rango Privadas	Máscara
A	0.0.0.0 a 127.255.255.255	10.0.0.0	255.0.0.0 = /8
B	128.0.0.0 a 191.255.255.255	172.16.0.0 a 172.31.0.0	255.255.0.0 = /16
C	192.0.0.0 a 223.255.255.255	192.168.0.0 a 192.168.255.0	255.255.255.0 = /24
D	224.0.0.0 a 239.255.255.255		
E	240.0.0.0 a 255.255.255.255		

Firstly, every single IP (direction in a network) in a ipv4 configuration belongs to a different class, which are: Class A, Class B, Class C, Class D and Class E. Furthermore, is accounted for 4GB of networks in total, delivering 2GB for Class A, 1GB for Class B, 512MB for Class C, and 256MB for Class D and Class E. Only from Class A to Class c are public and for general use, on the other hand, Class D is used for multicast and Class D for experimental use. Lastly, an IP is conformed by 32 bits, divided into bytes, and only separated with '.'.

An IP has a network part and a host part, that determines how many networks you can find in a class with their respective size and number of hosts. In a computer, this is described by the netmask which every IP has.

A netmask is conformed by '1's and '0's. The '1's are the network part and the '0's are the host part. To add, a netmask is really helpful for computers to know where the network part begins and ends, therefore with the host part can be done either. Specifically, the netmasks cannot have '0's and '1's alternatively, first are the ones and then the ceros.

A network which belongs to any Class is conformed by a network IP, a broadcast IP and a range of hosts. There is a simple way to get the network IP and the broadcast IP, using the logic operator "or" and "and" to get each. In addition, a network IP and a broadcast can not be gotten in a Class D or Class E.

The program "IP-Calculator" developed in language C, gets an IP (adress) and returns it's class, the network IP, the broadcast IP and the hosts range. In the end, the program asks if another IP will be written in to be processed.

Bits Operators

"Motion operators are operators at the bit level, and what they do is convert a certain amount to its equivalent in bits to later perform a displacement of that value. These operators are:

Nombre del operador	Sintaxis
Desplazamiento a la izquierda	a << b
Asignación con desplazamiento a la izquierda	a <<= b
Desplazamiento a la derecha	a >> b
Asignación con desplazamiento a la derecha	a >>= b
Complemento a uno	~a
AND binario	a & b
Asignación con AND binario	a &= b
OR binario	a | b
Asignación con OR binario	a |= b
XOR binario	a ^ b
Asignación con XOR binario	a ^= b

I will explain the use of displacement since at the beginning of the practice they were useful.

Despite being "operators for handling bits", all of them require operands of type integer, which can be of any of its variants (short, long, signed or unsigned) and enumerations. That is, the starting material is bytes, one or several, depending on the type of integer used.

If the operands are not integers, the compiler performs the relevant conversion, so the result is always an integer of the same type as the operands.

Do not confuse the bit operators, & and |, with the logical operators && and ||. Regarding the treatment of the sign, &, >>, << are sensitive to the context.
& can also be the reference operator of pointers, and reference declarator.

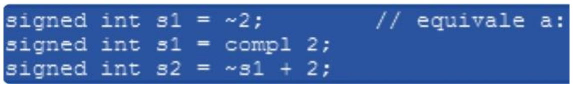
The Standard C ++ library has overloaded the << and >> operators for the basic types, so they can be used as output and input operators.

The result of the AND, XOR and OR operators is independent of the order of placement of their operands. The operators that enjoy this property are called associative. It is equivalent to the commutative property of certain arithmetic operators

.

Complement

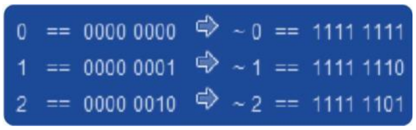
The complement operator is the only unary operator in terms of bit handling, and basically inverts each bit of the operand; 0 is converted into 1 and vice versa. It is also possible to use its functionality through the reserved word compl.



```
signed int s1 = ~2;        // equivale a:
signed int s1 = compl 2;
signed int s2 = ~s1 + 2;
```

Note that the result of this operator changes the sign of the operand, hence the "signed".

The binary representation of the complements to one of decimals 0, 1 and 2 are the ones that are expressed (to simplify we represent them as an octet):



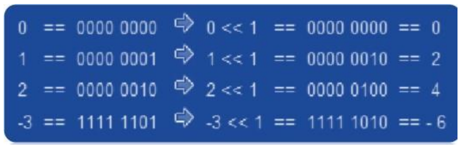
```
0  == 0000 0000  ⇨  ~ 0  == 1111 1111
1  == 0000 0001  ⇨  ~ 1  == 1111 1110
2  == 0000 0010  ⇨  ~ 2  == 1111 1101
```

Left Scrolling

This binary operator performs a bit shift to the left. The most significant bit (leftmost) is lost, and a least significant 0 is assigned (the one on the right). The right operand indicates the number of trips that will be made.

Recall that displacements are not rotations; bits that come from the left are lost, those that enter from the right are filled with zeros. This type of displacement is called logical as opposed to cyclic or rotational.

Example:



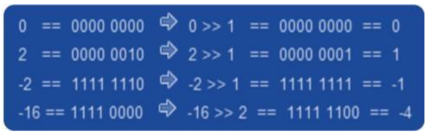
```
0  == 0000 0000  ⇨  0 << 1   == 0000 0000 == 0
1  == 0000 0001  ⇨  1 << 1   == 0000 0010 == 2
2  == 0000 0010  ⇨  2 << 1   == 0000 0100 == 4
-3 == 1111 1101  ⇨  -3 << 1  == 1111 1010 == - 6
```

As is well known, the unit displacement to the left is equivalent to multiplying the value of the displaced operand by two.

Right Scrolling

The least significant bit (to the right) is lost, but it must be noted that if the displaced expression is a signed integer and is negative, the result depends on the implementation. In addition, it is necessary to emphasize that as with the right shift, the second operand or displacement factor must be positive and of shorter length than that of the first operand.

Otherwise, the behavior of this operator is analogous to the previous one (left shift). For example:

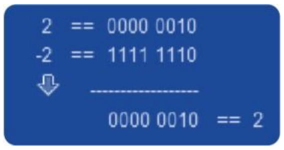


```
0   == 0000 0000  ⇨  0 >> 1    == 0000 0000  == 0
2   == 0000 0010  ⇨  2 >> 1    == 0000 0001  == 1
-2  == 1111 1110  ⇨  -2 >> 1   == 1111 1111  == -1
-16 == 1111 0000  ⇨  -16 >> 2  == 1111 1100  == -4
```

In contrast to the left shift, the unit shift to the right is equivalent to dividing the first operand by 2.

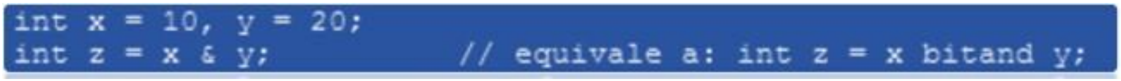
AND

This binary operator compares both operands bit by bit, and as a result returns a value constructed in such a way, that each bit is 1 if the corresponding bits of the operands are at 1. Otherwise, the bit is 0:



```
 2  == 0000 0010
-2  == 1111 1110
⇩      -----------------
       0000 0010  == 2
```

It is also possible to use the functionality of the operator "&" through the reserved word bitand, as shown below:

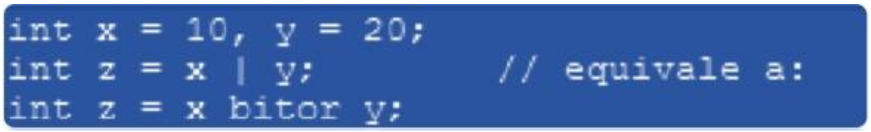


```
int x = 10, y = 20;
int z = x & y;          // equivale a: int z = x bitand y;
```

In some compilers the words reserved as bitand or compl can be deactivated or without support so it is advisable to always use the operator.

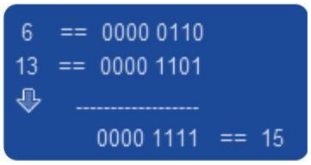
OR

This binary operator has an operation similar to the AND operator, except that in this case the result is 1 if any of them is 1. If not, it returns 0 (see example). You can use the word bitor to replace the operator "|":



```
int x = 10, y = 20;
int z = x | y;          // equivale a:
int z = x bitor y;
```

Example:



```
6   == 0000 0110
13  == 0000 1101
⇩      -----------------
       0000 1111  == 15
```

Software (libraries, packages, tools):

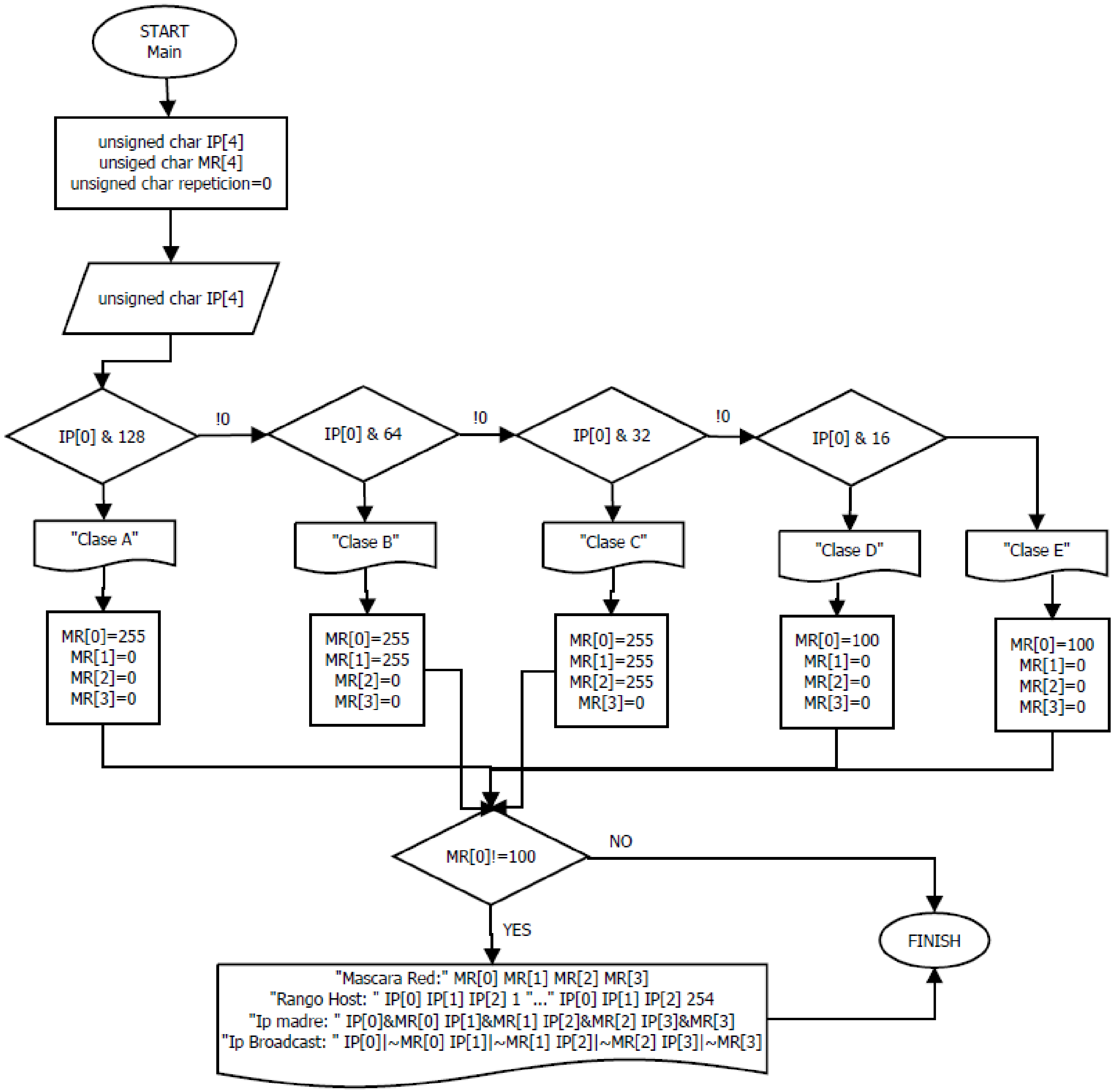
*Language C

Libraries: #include<studio.h>

Compilator: GCC Compilator

Software: text-editor and windows cmd.

Procedure:



Procedure:

1. The software shows the string “Hola Nidia /n Computadora: 19 /n IP:”
2. The program requests an IP
3. The program makes an AND operation with the first MSB, if the operation returns anything, is classified as a Class C. In case that the operation returns 0, another AND operation is done but now with the second MSB. The procedure is repeated until the fifth MSB is used.
4. Depending on the class, a netmask is assigned in the netmask array.
5. Once a class is defined and a netmask assigned, the mother IP and Broadcast is calculated.
6. An AND operation between the IP array and Netmask array, this is used to calculate Mother IP.
7. The netmask array is changed by a NOT netmask using ‘~’.
8. An OR operation between the IP array and Not Netmask Array is done to get the broadcast IP.
9. The range of hosts is gotten with the IP array, only the last space of the array is used. To get the first number a 1 is added, and the last number is gotten adding 254 to the number.
10. Lastly, the program prints out the results and ask the user if another IP will be put.

Results

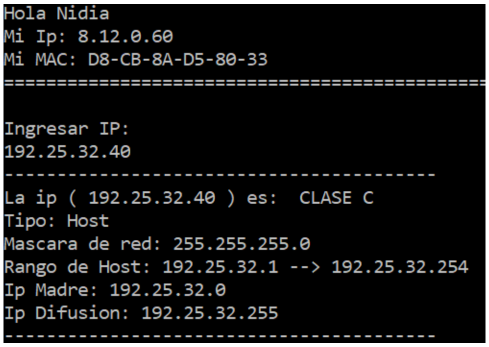
0000 0000	IP[0]
0000 0000	IP[1]
0000 0000	IP[2]
0000 0000	IP[3]
0000 0000	MR[0]
0000 0000	MR[1]
0000 0000	MR[2]
0000 0000	MR[3]
0000 0000	Repetición

192.25.32.40

1100 0100	IP[0]
0001 1010	IP[1]
0010 0000	IP[2]
0010 1000	IP[3]
0000 0000	MR[0]
0000 0000	MR[1]
0000 0000	MR[2]
1111 1111	MR[3]
0000 0000	Repetición

Memory map before initializing the program processes.

Map memory after making al processes in the program



```
Hola Nidia
Mi Ip: 8.12.0.60
Mi MAC: D8-CB-8A-D5-80-33
=========================================

Ingresar IP:
192.25.32.40
----------------------------------------
La ip ( 192.25.32.40 ) es:  CLASE C
Tipo: Host
Mascara de red: 255.255.255.0
Rango de Host: 192.25.32.1 --> 192.25.32.254
Ip Madre: 192.25.32.0
Ip Difusion: 192.25.32.255
----------------------------------------
```

FIGURE 1:Ip ingresada y como resultado, su ip madre, ip broadcast, rango de host

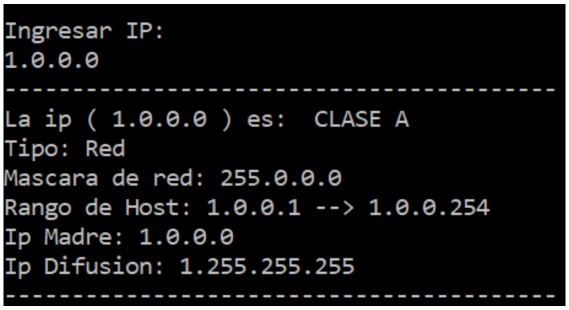
0000 0000	IP[0]
0000 0000	IP[1]
0000 0000	IP[2]
0000 0000	IP[3]
0000 0000	MR[0]
0000 0000	MR[1]
0000 0000	MR[2]
0000 0000	MR[3]
0000 0000	Repetición

1.0.0.0

0000 0001	IP[0]
0000 0000	IP[1]
0000 0000	IP[2]
0000 0000	IP[3]
0000 0000	MR[0]
1111 1111	MR[1]
1111 1111	MR[2]
1111 1111	MR[3]
0000 0000	Repetición

Memory map before initializing the program processes.

Map memory after making al processes in the program



```
Ingresar IP:
1.0.0.0
---------------------------------------
La ip ( 1.0.0.0 ) es:  CLASE A
Tipo: Red
Mascara de red: 255.0.0.0
Rango de Host: 1.0.0.1 --> 1.0.0.254
Ip Madre: 1.0.0.0
Ip Difusion: 1.255.255.255
---------------------------------------
```

FIGURE 2: Ip ingresada y como resultado, su ip madre, ip broadcast, rango de host

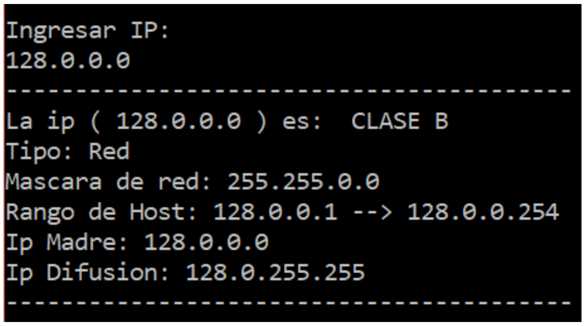
0000 0000	IP[0]
0000 0000	IP[1]
0000 0000	IP[2]
0000 0000	IP[3]
0000 0000	MR[0]
0000 0000	MR[1]
0000 0000	MR[2]
0000 0000	MR[3]
0000 0000	Repetición

128.0.0.0

1000 0000	IP[0]
0000 0000	IP[1]
0000 0000	IP[2]
0000 0000	IP[3]
0000 0000	MR[0]
0000 0000	MR[1]
1111 1111	MR[2]
1111 1111	MR[3]
0000 0000	Repetición

Memory map before initializing the program processes.

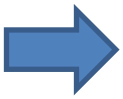
Map memory after making al processes in the program



```
Ingresar IP:
128.0.0.0
----------------------------------------
La ip ( 128.0.0.0 ) es:  CLASE B
Tipo: Red
Mascara de red: 255.255.0.0
Rango de Host: 128.0.0.1 --> 128.0.0.254
Ip Madre: 128.0.0.0
Ip Difusion: 128.0.255.255
----------------------------------------
```

FIGURE 3: Ip ingresada y como resultado, su ip madre, ip broadcast, rango de host

0000 0000	IP[0]
0000 0000	IP[1]
0000 0000	IP[2]
0000 0000	IP[3]
0000 0000	MR[0]
0000 0000	MR[1]
0000 0000	MR[2]
0000 0000	MR[3]
0000 0000	Repetición

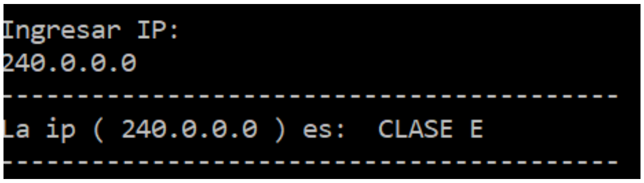


240.0.0.0

1110 0000	IP[0]
0000 0000	IP[1]
0000 0000	IP[2]
0000 0000	IP[3]
0110 0100	MR[0]
0000 0000	MR[1]
0000 0000	MR[2]
0000 0000	MR[3]
0000 0000	Repetición

Memory map before initializing the program processes.

Map memory after making al processes in the program



```
Ingresar IP:
240.0.0.0
----------------------------------------
La ip ( 240.0.0.0 ) es:  CLASE E
----------------------------------------
```

FIGURE 4: Resultado de ingresar una Ip clase E.

NOTE: The same happens if a Class D Ip is written.

Discussion:

The results show that the any network IP and any broadcast IP can be calculated with a simple logic operation, furthermore, no memory is wasted and unnecessary computer processing is used.

The results are of great importance, also show the use and manage of bits in order to obtain different information by only introducing an IP address.

In the main structure we will place all our code from the ip validation, to return the range of hosts. The important thing to ignore in the validation of ip is that by not using a repetitive structure (for), we do not define the maximum range of octets that the user can enter, therefore you can enter as many as you want and the program will not mark an error, however we will not take them into account for further development of the program. Recall that the task of the repetitive structure "for" is to make a jump of size 'n' forward or backward delimited within a range.

Conclusions:

All in all, this practice was helpful in order to learn how a logic operation works, furthermore how to use the minimum storage in the memory, making a faster and more optimized program. Firstly, all learnt can be used for programming, taking in account how much memory resource is needed when a software is developed.

Secondly, some errors which could have occurred were in the usage of the operation NOT, because it had to be done in a separated line and not into a print function, this would take different bits which we needed.

At last, the results applied generally quite well, showing all the main requested results. In my opinion, There were not design inconsistencies, from the use of memory, to the processing management.

References:

Cortes Duarte, N. (2018). Class Summary. [Text] Instituto Politecnico Nacional ESCOM, Mexico City.

Ecured.cu. (n.d.). Operadores lógicos - EcuRed. [online] Available at: https://www.ecured.cu/Operadores_l%C3%B3gicos [Accessed 22 Sep. 2018].

Avella, R. (2018). stdio.h y sus funciones - El blog de rikrdoavella.over-blog.es. [online] El blog de rikrdoavella.over-blog.es. Available at: http://rikrdoavella.over-blog.es/article-stdio-h-y-sus-funciones-55958648.html [Accessed 22 Sep. 2018].

Code:

```c
#include <stdio.h>
#include <iso646.h>

void main (void){
unsigned char ip[4];
unsigned char mascarared[4];
unsigned char repeticion=0;

    printf("Hola Nidia\n");
    printf("Mi Ip: 8.12.0.60\n");
    printf("Mi MAC: D8-CB-8A-D5-80-33");

    do{
        printf("\n======================================================");
        repeticion=0;
        printf("\n\nIngresar IP: \n");
        scanf("%d.%d.%d.%d",&ip[0],&ip[1],&ip[2],&ip[3]);
        printf("-----------------------------------------\n");
        printf("La ip ( %d.%d.%d.%d ) es:  ",ip[0],ip[1],ip[2],ip[3]);

            if (ip[0] & 128){
                if(ip[0] & 64){
                    if(ip[0] & 32){
                        if (ip[0] & 16){
                            printf("CLASE E");
                            mascarared[0]=100;
                            mascarared[1]=0;
                            mascarared[2]=0;
                            mascarared[3]=0;
                        }else{
                            printf("CLASE D");
                            mascarared[0]=100;
                            mascarared[1]=0;
                            mascarared[2]=0;
                            mascarared[3]=0;
                        }
                    }else{
                        printf("CLASE C");
                        mascarared[0]=255;
                        mascarared[1]=255;
                        mascarared[2]=255;
                        mascarared[3]=0;
                        //------------------------------------------ Encontrar su tipo
                        if (ip[3] | 0){
                            if ((ip[3] | 0)==255){
                                printf("\nTipo: Broadcast");
                            }else{
                                printf("\nTipo: Host");

                            }
                        }else{
                            printf("\nTipo: Red");
                        }
                    }
                }else{
                    printf("CLASE B");
                    mascarared[0]=255;
                    mascarared[1]=255;
                    mascarared[2]=0;
                    mascarared[3]=0;
                    //------------------------------------------Encontrar su tipo
                    if ((ip[3] | 0)||(ip[2] | 0)){
                        if (((ip[3] | 0)==255)&&((ip[2] | 0)==255)){
                            printf("\nTipo: Broadcast");
                        }else{
                            printf("\nTipo: Host");

                        }
                    }else{
                        printf("\nTipo: Red");
                    }
                }
            }else{
                printf("CLASE A");
                mascarared[0]=255;
                mascarared[1]=0;
                mascarared[2]=0;
                mascarared[3]=0;
                //------------------------------------------Encontrar su tipo
                if ((ip[3] | 0)||(ip[2] | 0)||(ip[1] | 0)){
                        if (((ip[3] | 0)==255)&&((ip[2] | 0)==255)&&((ip[1] | 0)==255)){
                            printf("\nTipo: Broadcast");
                        }else{
                            printf("\nTipo: Host");

                        }
                    }else{
                        printf("\nTipo: Red");
                    }
            }
/**---------------------------------------------------------------Obtenemos los demas valores que son solicitados para la red*/
        if (mascarared[0]!=100){
            printf("\nMascara de red: %d.%d.%d.%d",mascarared[0],mascarared[1],mascarared[2],mascarared[3]);
            printf("\nRango de Host: %d.%d.%d.%d",(ip[0]),(ip[1]),(ip[2]),1);
            printf(" --> %d.%d.%d.%d",(ip[0]),(ip[1]),(ip[2]),254);
            printf("\nIp Madre: %d.%d.%d.%d",(ip[0]&mascarared[0]),(ip[1]&mascarared[1]),(ip[2]&mascarared[2]),(ip[3]&mascarared[3]));
            mascarared[0]=~mascarared[0];
            mascarared[1]=~mascarared[1];
            mascarared[2]=~mascarared[2];
                  mascarared[3]=~mascarared[3];
                  printf("\nIp Difusion: %d.%d.%d.%d",(ip[0]|(mascarared[0])),(ip[1]|(mascarared[1])),(ip[2]|(mascarared[2])),(ip[3]|(mascarared[3])));

              }

      /**-----------------------------------------------------------------Preguntamos al usuario si quiere repetir el proceso*/
              printf("\n-----------------------------------------");
              printf("\nDesea Ingresar otra IP? (1/si)(2/no): ");
              scanf("%d",&repeticion);
              system ("cls");
          }while(repeticion!=2);
        printf("\nFIN DEL PROGRAMA");
    }
```