CRC with error checking (Cyclic Redundancy Check):

def xor(a, b):

result = []

for i in range(1, len(b)):

if a[i] == b[i]:

result.append('0')

else:

result.append('1')

return ''.join(result)

def mod2div(dividend, divisor):

flag = len(divisor)

tmp = dividend[0: flag]

while flag < len(dividend):

if tmp[0] == '1':

tmp = xor(divisor, tmp) + dividend[flag]

else:

tmp = xor('0'\*flag, tmp) + dividend[flag]

flag += 1

if tmp[0] == '1':

tmp = xor(divisor, tmp)

else:

tmp = xor('0'\*flag, tmp)

return tmp

def encodeData(data, key):

l\_key = len(key)

appended\_data = data + '0'\*(l\_key-1)

remainder = mod2div(appended\_data, key)

codeword = data + remainder

return codeword

def checkError(received\_data, key):

remainder = mod2div(received\_data, key)

return remainder, remainder == '0' \* (len(key) - 1)

def binary\_to\_int(binary\_str):

return int(binary\_str, 2) if binary\_str else 0

if \_\_name\_\_ == "\_\_main\_\_":

print("Sender's side : ")

data = input("Enter the data: ")

key = input("Enter the key: ")

encoded\_data = encodeData(data, key)

print("Initial Dataword (Dividend) : " ,(data+'000'))

print("Encoded Data (Data + Remainder) of sender :", encoded\_data)

print(" ")

print("Receiver's side : ")

received\_data = input("Enter the received data: ")

remainder, is\_check= checkError(received\_data, key)

print("Remainder of received data :", remainder)

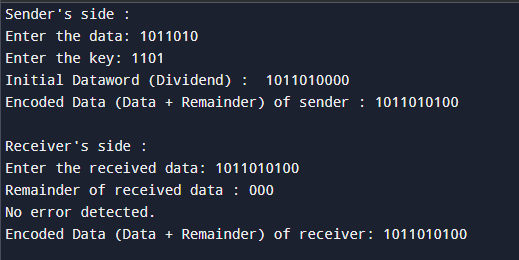
if is\_check:

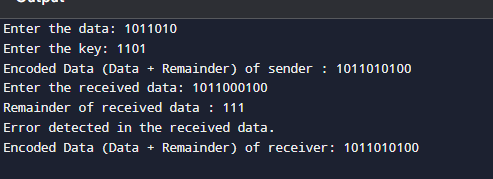
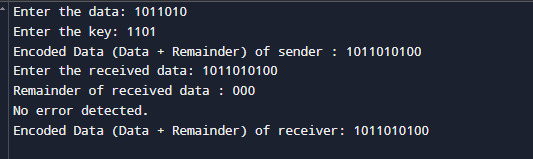
print("No error detected.")

print("Encoded Data (Data + Remainder) of receiver:", encoded\_data)

else:

print("Error detected in the received data.")

****

**  
**

Internet CheckSum:

def calculate\_checksum(data):

words = [int(data[i:i+4], 16) for i in range(0, len(data), 4)]

sum\_value = sum(words)

sum\_value = (sum\_value & 0xFFFF) + (sum\_value >> 16)

checksum = ~sum\_value & 0xFFFF

return format(checksum, '04X')

def verify\_checksum(data):

words = [int(data[i:i+4], 16) for i in range(0, len(data), 4)]

sum\_value = sum(words)

sum\_value = (sum\_value & 0xFFFF) + (sum\_value >> 16)

return (sum\_value & 0xFFFF) == 0xFFFF

def print\_words(data):

print("16-bit Words:")

for i in range(0, len(data), 4):

print(data[i:i+4])

def main():

print("Sender's side:")

data = input("Enter the hexadecimal data (without spaces): ").strip().upper()

if len(data) % 4 != 0:

data = data.zfill(len(data) + (4 - len(data) % 4))

print\_words(data)

sum\_value = sum(int(data[i:i+4], 16) for i in range(0, len(data), 4))

print(f"Sum of 16-bit words: {format(sum\_value, '04X')}")

sum\_value = (sum\_value & 0xFFFF) + (sum\_value >> 16)

print(f"Adjusted Sum after adding carry: {format(sum\_value, '04X')}")

checksum = calculate\_checksum(data)

print(f"Calculated Checksum: {checksum}")

full\_packet = data + checksum

print(f"\nFull Packet: {full\_packet}")

print("\nReceiver's side:")

received\_packet = input("Enter the received packet (without spaces): ").strip().upper()

print\_words(received\_packet)

sum\_value = sum(int(received\_packet[i:i+4], 16) for i in range(0, len(received\_packet), 4))

sum\_value = (sum\_value & 0xFFFF) + (sum\_value >> 16)

final\_result = ~sum\_value & 0xFFFF

print(f"1's Complement: {format(final\_result, '04X')}")

if verify\_checksum(received\_packet):

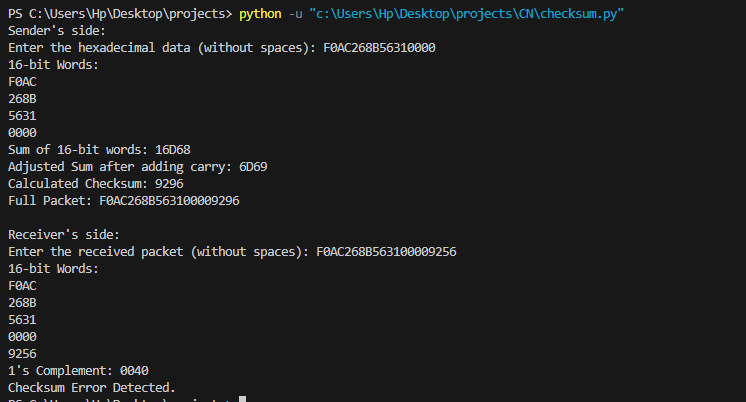
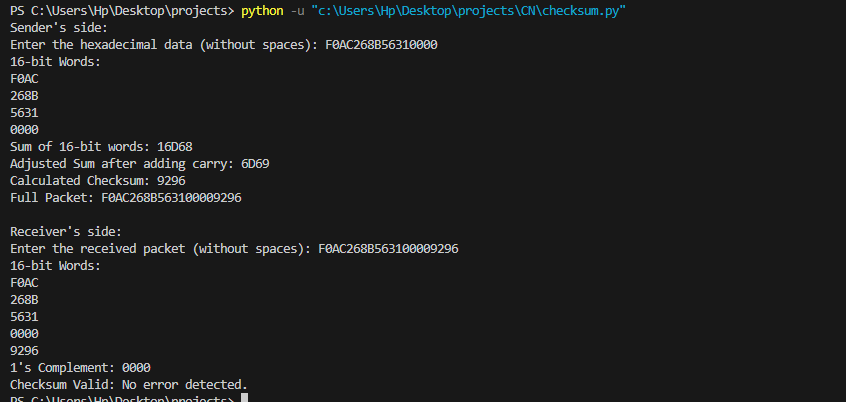
print("Checksum Valid: No error detected.")

else:

print("Checksum Error Detected.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

  
  
  
  
  
CLASSLESS AND CLASSFUL:  
#include <iostream>

#include <string>

using namespace std;

bool isValidIP(const string& ip) {

int octetCount = 0;

int currentOctet = 0;

bool isDigit = false;

for (char c : ip) {

if (c == '.') {

if (currentOctet > 255 || currentOctet < 0) return false;

currentOctet = 0;

++octetCount;

isDigit = false;

} else if (isdigit(c)) {

currentOctet = currentOctet \* 10 + (c - '0');

isDigit = true;

} else {

return false;

}

}

if (!isDigit || currentOctet > 255 || octetCount != 3) return false;

return true;

}

void classless() {

}

void classfull() {

string ip;

cout << "Enter IP Address: ";

cin >> ip;

if (!isValidIP(ip)) {

cout << "Invalid IP address format." << endl;

return;

}

int firstOctet = 0;

size\_t pos = ip.find('.');

if (pos != string::npos) {

firstOctet = 0;

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

firstOctet = firstOctet \* 10 + (ip[i] - '0');

}

}

string nwclass;

string mask;

if (firstOctet >= 1 && firstOctet <= 126) {

nwclass = "A";

mask = "255.0.0.0";

}

else if (firstOctet >= 128 && firstOctet <= 191) {

nwclass = "B";

mask = "255.255.0.0";

}

else if (firstOctet >= 192 && firstOctet <= 223) {

nwclass = "C";

mask = "255.255.255.0";

}

else {

nwclass = "Reserved";

mask = "-";

}

cout << "IP Address: " << ip << endl;

cout << "Class: " << nwclass << endl;

cout << "Default Mask: " << mask << endl;

}

int main() {

int choice;

cout << "Menu:\n";

cout << "1. Classful Addressing" << endl;

cout << "2. Classless Addressing" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Classful Addressing" << endl;

classfull();

break;

case 2:

cout << "Classless Addressing" << endl;

classless();

break;

default:

cout << "Invalid choice." << endl;

break;

}

return 0;

}

Expt 6

*#include* <iostream>

*#include* <cmath>

*#include* <vector>

using namespace std;

vector<int> convertToBinary(int *a*, int *b*, int *c*, int *d*)

{

   vector<int> binIp;

   int octets[4] = {*a*, *b*, *c*, *d*};

*for* (int i = 0; i < 4; ++i)

   {

*for* (int j = 7; j >= 0; --j)

         binIp.push\_back((octets[i] >> j) & 1);

   }

*return* binIp;

}

void displayClassfulAddress(int *a*, int *b*, int *c*, int *d*)

{

   string ip = to\_string(*a*) + "." + to\_string(*b*) + "." + to\_string(*c*) + "." + to\_string(*d*);

   int firstOctet = *a*;

   string ipClass, defaultMask;

*if* (firstOctet >= 1 && firstOctet <= 126)

   {

      ipClass = "A";

      defaultMask = "Default Mask: 255.0.0.0";

   }

*else* *if* (firstOctet >= 128 && firstOctet <= 191)

   {

      ipClass = "B";

      defaultMask = "Default Mask: 255.255.0.0";

   }

*else* *if* (firstOctet >= 192 && firstOctet <= 223)

   {

      ipClass = "C";

      defaultMask = "Default Mask: 255.255.255.0";

   }

*else* *if* (firstOctet >= 224 && firstOctet <= 239)

   {

      ipClass = "D";

      defaultMask = "Reserved for Multicasting (Class D)";

   }

*else* *if* (firstOctet >= 240 && firstOctet <= 255)

   {

      ipClass = "E";

      defaultMask = "Reserved for Research (Class E)";

   }

*else*

   {

      ipClass = "Invalid";

      defaultMask = "Invalid";

   }

   cout << "IP: " << ip << "\nIP Class: " << ipClass << "\n"

        << defaultMask << endl;

}

void classlessAddressing(int *a*, int *b*, int *c*, int *d*, int *cidrMask*)

{

   string ip = to\_string(*a*) + "." + to\_string(*b*) + "." + to\_string(*c*) + "." + to\_string(*d*);

   int hostBits = 32 - *cidrMask*;

   int totalHosts = pow(2, hostBits) - 2;

   cout << "\nClassless Addressing\nIP: " << ip << "\nCIDR Mask: /" << *cidrMask* << endl;

   cout << "Total Hosts: " << totalHosts << endl;

}

void calculateNetworkAddress(int *ip*[], int *mask*, vector<int> &*network*)

{

*for* (int i = 0; i < 4; ++i)

   {

*network*[i] = *ip*[i] & (*mask* >> (i \* 8) & 0xFF);

   }

}

void displaySubnetsWithLogic(int *ip*[], int *n*, int *setBits*)

{

   int num = 1;

*while* (num < *n*)

      num \*= 2;

   int bits = log2(num);

   int tot = (*setBits* + bits);

   int copy = 32 - tot;

   vector<int> subnet\_mask(4);

   vector<int> network(4);

*for* (int i = 0; i < 4; i++)

   {

      int a = 0;

*for* (int j = 7; j >= 0; j--)

      {

*if* (tot > 0)

         {

            a = (a | (1 << j));

            tot--;

         }

      }

      subnet\_mask[i] = a;

   }

   calculateNetworkAddress(*ip*, subnet\_mask[0] | (subnet\_mask[1] << 8) | (subnet\_mask[2] << 16) | (subnet\_mask[3] << 24), network);

   cout << "The IP is: " << *ip*[0] << "." << *ip*[1] << "." << *ip*[2] << "." << *ip*[3] << endl;

   cout << "The Subnet Mask is: " << subnet\_mask[0] << "." << subnet\_mask[1] << "." << subnet\_mask[2] << "." << subnet\_mask[3] << endl;

   cout << "Network Address: " << network[0] << "." << network[1] << "." << network[2] << "." << network[3] << endl;

   int amount = pow(2, copy);

   cout << "\n";

   cout << "The number of addresses per subnet: " << amount << endl;

   cout << "The IP Addresses are: " << endl;

*for* (int i = 0; i < num; i++)

   {

      cout << "\n";

      cout << "Block " << i + 1 << endl;

      cout << "Start: " << network[0] << "." << network[1] << "." << network[2] << "." << network[3] << endl;

      network[3] += amount - 1;

*for* (int j = 3; j >= 1; j--)

      {

*while* (network[j] > 255)

         {

            network[j] -= 256;

            network[j - 1] += 1;

         }

      }

      cout << "End: " << network[0] << "." << network[1] << "." << network[2] << "." << network[3] << endl;

      network[3]++;

*for* (int j = 3; j >= 1; j--)

      {

*while* (network[j] > 255)

         {

            network[j] -= 256;

            network[j - 1] += 1;

         }

      }

   }

}

int main()

{

   int choice, a, b, c, d, subnetCount, setBits;

   int ip[4];

   cout << "Enter the first octet (a): ";

   cin >> a;

   cout << "Enter the second octet (b): ";

   cin >> b;

   cout << "Enter the third octet (c): ";

   cin >> c;

   cout << "Enter the fourth octet (d): ";

   cin >> d;

   ip[0] = a;

   ip[1] = b;

   ip[2] = c;

   ip[3] = d;

*while* (true)

   {

      cout << "Choose addressing method:\n";

      cout << "1. Classful Addressing\n";

      cout << "2. Classless Addressing (CIDR)\n";

      cout << "3. Subnetting\n";

      cout << "4. Exit\n";

      cout << "Enter your choice: ";

      cout << "\n";

      cin >> choice;

*switch* (choice)

      {

*case* 1:

         displayClassfulAddress(a, b, c, d);

         cout << "\n";

*break*;

*case* 2:

         cout << "Enter CIDR mask: ";

         cin >> setBits;

         classlessAddressing(a, b, c, d, setBits);

         cout << "\n";

*break*;

*case* 3:

         cout << "\n";

         cout << "Enter number of subnets: ";

         cin >> subnetCount;

         cout << "Enter number of set bits for subnet mask: ";

         cin >> setBits;

         cout << "\n";

         displaySubnetsWithLogic(ip, subnetCount, setBits);

         cout << "\n";

*break*;

*case* 4:

         cout << "Exiting program." << endl;

*return* 0;

*default*:

         cout << "Invalid choice." << endl;

*break*;

      }

   }

*return* 0;

}