Computer Networking lab

1. **WAP to design CRC encoder and decoder and check for errors.**

```
#include <stdio.h>
#include <string.h>
void xorOperation(char *crc, char *divisor, int len) {
   for(int i = 0; i < len; i++) {
      if(crc[i] == divisor[i])
         crc[i] = '0';
      else
         crc[i] = '1';
   }
}
void crc(char *data, char *divisor, char *remainder) {
   int data_len = strlen(data);
   int divisor_len = strlen(divisor);
   for(int i = 0; i < divisor_len - 1; i++)
      data[data_len + i] = '0';
   for(int i = 0; i <= data_len; i++) {
      if(data[i] == '1')
         xorOperation(&data[i], divisor, divisor_len);
   }
   strncpy(remainder, &data[data_len], divisor_len - 1);
   remainder[divisor_len - 1] = '\0';
}

int main() {
   char data[100], divisor[30], remainder[30];

   printf("Enter the data: ");
   scanf("%s", data);

   char copyData[100];
   strcpy(copyData, data);

   printf("Enter the divisor: ");
   scanf("%s", divisor);

   crc(data, divisor, remainder);

   printf("The remainder is: %s\n", remainder);

   printf("The codeword is: %s%s\n", copyData, remainder);

   strcat(copyData, remainder);

   char receivedData[100];
   printf("Enter the received data: ");
   scanf("%s", receivedData);

   if(strcmp(receivedData, copyData) == 0)
```

```
    printf("No error in the transmitted data.\n");
  else
    printf("Error in the transmitted data.\n");

  return 0;
}
```

2. WAP to design a single bit parity check code.

```
#include <bits/stdc++.h>
using namespace std;

int main(){
  int n;
  cout <<"Enter the number of bits: ";
  cin >> n;
  vector <int> input_bits(n);
  cout << "Enter the bits: "<< endl;
  for(int i = 0; i < n; i++)
    cin >> input_bits[i];
  vector <int> repetition_code;
  for(auto it: input_bits){
    repetition_code.push_back(it);
    repetition_code.push_back(it);
  }
  cout << "Input bits: ";
  for(auto it: input_bits)
    cout << it;
  cout << endl;
  cout << "Repetition code: ";
  for(auto it: repetition_code)
    cout << it;
  cout << endl;
  cout << "Enter the number of bits of the data word: ";
  cin >> n;
  vector <int> code(n);
  cout << "Enter the data word: "<< endl;
  for(int i = 0; i < n; i++){
    int bit;
    cin >> bit;
    code.push_back(bit);
  }
  int result = 0;
  for(int i = 0; i < repetition_code.size(); i++){
    int intermediate = 0;
    intermediate = intermediate^repetition_code[i];
    if(i%2 == 0){
      result = result^intermediate;
    }
  }
  for(int i = 0; i < code.size(); i++){
    int intermediate = 0;
```

```
      intermediate = intermediate^code[i];
      result = result^intermediate;
   }
   if(result != 0)
      cout << "Not valid. Error detected." << endl;
   else
      cout << "Valid. No error detected." << endl;
   return 0;
}
```

3. WAP to design a double bit parity check code.

```
#include <iostream>
#include <bitset>
int main() {
   std::string dataword;
   std::cout << "Enter a 2-bit dataword: ";
   std::cin >> dataword;
   if(dataword.length() != 2) {
      std::cout << "Error: Invalid input. Please enter a 2-bit dataword.\n";
      return -1;
   }
   std::string codeword = dataword + dataword;

   int xor1 = (codeword[0] - '0') ^ (codeword[1] - '0');
   int xor2 = (codeword[2] - '0') ^ (codeword[3] - '0');
   int xorFinal = xor1 ^ xor2;
   if(xorFinal == 0) {
      codeword += '0';
      std::cout << "Error condition met. The final codeword is: " << codeword << "\n";
   } else {
      codeword += std::to_string(xorFinal);
      std::cout << "The final codeword is: " << codeword << "\n";
   }

   return 0;
}
```

4. WAP to calculate hamming distance and minimum hamming distance, s and t.

```
#include<bits/stdc++.h>
using namespace std;

string xorOp(string a, string b, int n){
        string c = "";
        for (int i = 0; i < n; i++)
        {
                if (a[i] == b[i])
                        c += '0';
                else
                        c += '1';
```

```
        }
        return c;
}

int main()
{
        string a = "01011";
        string b = "10101";
        string c = "11000";
        string x = "00000";

        int n = a.length();

        string dA = xorOp(x, a, n);
        string dB = xorOp(x, b, n);
        string dC = xorOp(x, c, n);
        int countA =0;
        int countB =0;
        int countC =0;



        for(auto i=0;i<n;i++){
                if(dA[i]=='1'){
                        countA++;
                }
                if(dB[i]=='1'){
                        countB++;
                }
                if(dC[i]=='1'){
                        countC++;
                }
        }

        cout << "hD of dA= "<< countA<<endl;
        cout << "hD of dB= "<< countB<<endl;
        cout << "hD of dC= "<< countC<<endl;

        int dMin=min(countA,min(countB,countC));

        cout <<"\n"<<dMin<<" bits are minimum"<<endl;

        cout << "min no of errors can be detected (s)=" << dMin-1<<endl;

        int t=(dMin-1)/2;
        cout << "min no of errors can be corrected (t)=" << t<<endl;


}
```

5. WAP to identify the class of IPv4 address.

```
#include <stdio.h>
#include <string.h>

/*
Function : extractIpAddress
Arguments :
1) sourceString - String pointer that contains ip address
2) ipAddress - Target variable short type array pointer that will store ip address octets
*/
void extractIpAddress(unsigned char *sourceString,short *ipAddress)
{
   unsigned short len=0;
   unsigned char  oct[4]={0},cnt=0,cnt1=0,i,buf[5];

   len=strlen(sourceString);
   for(i=0;i<len;i++)
   {
      if(sourceString[i]!='.'){
         buf[cnt++] =sourceString[i];
      }
      if(sourceString[i]=='.' || i==len-1){
         buf[cnt]='\0';
         cnt=0;
         oct[cnt1++]=atoi(buf);
      }
   }
   ipAddress[0]=oct[0];
   ipAddress[1]=oct[1];
   ipAddress[2]=oct[2];
   ipAddress[3]=oct[3];
}

int main()
{
   unsigned char ip[20]={0};
   short ipAddress[4];

   printf("Enter IP Address (xxx.xxx.xxx.xxx format): ");
   scanf("%s",ip);

   extractIpAddress(ip,&ipAddress[0]);

   printf("\nIp Address: %03d. %03d. %03d. %03d\n",ipAddress[0],ipAddress[1],ipAddress[2],ipAddress[3]);

   if(ipAddress[0]>=0 && ipAddress[0]<=127)
      printf("Class A Ip Address.\n");
   if(ipAddress[0]>127 && ipAddress[0]<191)
      printf("Class B Ip Address.\n");
   if(ipAddress[0]>191 && ipAddress[0]<224)
      printf("Class C Ip Address.\n");
```

```
   if(ipAddress[0]>224 && ipAddress[0]<=239)
      printf("Class D Ip Address.\n");
   if(ipAddress[0]>239)
      printf("Class E Ip Address.\n");

   return 0;
}
```

6. WAP to print the first and last IP Address.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

int ind = 0;
int ia[36];

void extractIpAddress(unsigned char *sourceString, short *ipAddress)
{
  unsigned short len = 0;
  unsigned char oct[4] = {0}, cnt = 0, cnt1 = 0, i, buf[5];

  len = strlen(sourceString);
  for (i = 0; i < len; i++)
  {
    if (sourceString[i] != '.')
    {
      buf[cnt++] = sourceString[i];
    }
    if (sourceString[i] == '.' || i == len - 1)
    {
      buf[cnt] = '\0';
      cnt = 0;
      oct[cnt1++] = atoi(buf);
    }
  }
  ipAddress[0] = oct[0];
  ipAddress[1] = oct[1];
  ipAddress[2] = oct[2];
  ipAddress[3] = oct[3];
}
void bin(int numb, int m)
{
  int d = 0;
  int c = 7;
  int sum[] = {0, 0, 0, 0, 0, 0, 0, 0};

  while (numb > 0)
  {
    d = numb % 2;
```

```
    sum[c] = d;
    numb = numb / 2;
    c = c - 1;
  }
  sum[c] = numb;
  int sub = 32 - m;

  for (int i = sub; i < 8; i++)
  {
    sum[i] = 0;
  }
  printf("\n the first ipAddress ");
  for (int i = 0; i < ind; i++)
  {
    printf("%d", ia[i]);
    if (i % 8 == 0 && i != 0)
      printf(".");
  }
  printf(".");
  for (int i = 0; i < 8; i++)
  {
    printf("%d", sum[i]);
  }
  for (int i = sub; i < 8; i++)
  {
    sum[i] = 1;
  }
  printf("\n the last ipAddress ");
  for (int i = 0; i < ind; i++)
  {
    printf("%d", ia[i]);
    if (i % 8 == 0 && i != 0)
      printf(".");
  }
  printf(".");
  for (int i = 0; i < 8; i++)
  {
    printf("%d", sum[i]);
  }
}

void dtb(int numb)
{
  int d = 0;
  int c = 7;
  int sum[] = {0, 0, 0, 0, 0, 0, 0, 0};

  while (numb > 0)
  {
    d = numb % 2;
    sum[c] = d;
```

```
    numb = numb / 2;
    c = c - 1;
  }
  sum[c] = numb;
  for (int i = 0; i < 8; i++)
  {
    ia[ind++] = sum[i];
  }
}

int main()
{
  unsigned char ip[20] = {0};
  int m;
  short ipAddress[4];
  printf("Enter IP Address (xxx.xxx.xxx.xxx format): ");
  scanf("%s", ip);
  printf("Enter the mask address");
  scanf("%d", &m);
  extractIpAddress(ip, &ipAddress[0]);
  int n;
  for (int i = 0; i < 3; i++)
  {
    n = ipAddress[i];
    dtb(n);
  }
  n = ipAddress[3];
  bin(n, m);
  return 0;
}
```

7. WAP to print all the IP address for a given classless IP.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
int ind = 0;
int ia[36];
void print(int *a)
{
   for (int i = 0; i < 32; i++)
   {
      printf("%d", a[i]);
      if ((i + 1) % 8 == 0 && i != 0 && i != 31)
         printf(".");
   }
   printf("\n ");
}
void nextIpAddress(int m)
{
```

```
    int b[32] = {0};
    b[31] = 1;

    int c = 0;
    int s = 0;
    for (int i = 31; i >= 0; i--)
    {
        s = b[i] + ia[i] + c;
        c = s / 2;
        s = s % 2;

        ia[i] = s;
    }
}
void allIpAddress(int m)
{
    int sub = 32 - m;

    for (int i = m; i < 32; i++)
    {
        ia[i] = 0;
    }
    printf("All the ipAddress are : \n");
    print(ia);
    for (int i = 1; i < pow(2, sub); i++)
    {
        nextIpAddress(m);
        print(ia);
    }
}

void extractIpAddress(unsigned char *sourceString, short *ipAddress)
{
    unsigned short len = 0;
    unsigned char oct[4] = {0}, cnt = 0, cnt1 = 0, i, buf[5];

    len = strlen(sourceString);
    for (i = 0; i < len; i++)
    {
        if (sourceString[i] != '.')
        {
            buf[cnt++] = sourceString[i];
        }
        if (sourceString[i] == '.' || i == len - 1)
        {
            buf[cnt] = '\0';
            cnt = 0;
            oct[cnt1++] = atoi(buf);
        }
    }
    ipAddress[0] = oct[0];
```

```
   ipAddress[1] = oct[1];
   ipAddress[2] = oct[2];
   ipAddress[3] = oct[3];
}
void convertIpAddressToBinary(short *ipAd)
{
   int d = 0;
   int c = 7;
   int sum[] = {0, 0, 0, 0, 0, 0, 0, 0};
   int numb = 0;

   for (int i = 0; i < 4; i++)
   {
      numb = ipAd[i];

      while (numb > 0)
      {
         d = numb % 2;
         sum[c] = d;
         numb = numb / 2;
         c = c - 1;
      }
      sum[c] = numb;
      for (int i = 0; i < 8; i++)
      {
         ia[ind++] = sum[i];
         sum[i] = 0;
      }
      c = 7;
   }
}
int main()
{
   unsigned char ip[20] = {0};
   int m;
   short ipAddress[4];
   printf("Enter IP Address (xxx.xxx.xxx.xxx format): ");
   scanf("%s", ip);
   printf("Enter the mask address");
   scanf("%d", &m);

   extractIpAddress(ip, &ipAddress[0]);
   convertIpAddressToBinary(ipAddress);
   printf("The ipAddress in binary is : \n");
   print(ia);
   allIpAddress(m);
   return 0;
}
```

8. WAP for error detection using checksum(frame length=3).

```
#include <bits/stdc++.h>
```

```
using namespace std;
void decToBinary(int n,string &binaryNumber)
{
   int binaryNum[32];
   int i = 0;
   while (n > 0) {
      binaryNum[i] = n % 2;
      n = n / 2;
      i++;
   }
   for (int j = i - 1; j >= 0; j--){
      binaryNumber+= to_string(binaryNum[j]);
   }
}
void receiver(string &number){
    int remainder = number.length() % 3;
    if (remainder != 0) {
      number.append(3 - remainder, '0');
   }

   for (int i = 0; i < number.length(); i=i+3) {
      int num = 0;
      for (int j = 0; j < 3; j++) {
         num = num + (number[i + j] - '0') * pow(2, 2 - j);
      }
      cout << num <<" ";
   }
}
int main() {
   string binaryNumber;
   cout << "Enter a binary number: ";
   cin >> binaryNumber;
   int len=binaryNumber.length();

   int remainder = binaryNumber.length() % 3;
   if (remainder != 0) {
      binaryNumber.append(3 - remainder, '0');
   }

   int sum = 0;
   for (int i = 0; i < binaryNumber.length(); i=i+3) {
      int num = 0;
      for (int j = 0; j < 3; j++) {
         num = num + (binaryNumber[i + j] - '0') * pow(2, 2 - j);
      }
      sum += num;

   }
   cout << "Total sum: " << sum << endl;
   decToBinary(sum,binaryNumber);
   cout << binaryNumber<<endl;
```

```
    cout << "val: "<<sum<<endl;
    cout << binaryNumber.substr(0,len)<<endl;
    string num=binaryNumber.substr(0,len);
    receiver(num);
    return 0;
}
```

9. Write a program in c to implement row and column parity of a 2D matrix.

```
#include <iostream>
#include <vector>
using namespace std;
void display(vector<string> v)
{
   for (auto it : v)
   {
      cout << it << endl;
   }
}
void columnParity(vector<string> v)
{
   string cp = "";
   int k = v[0].length();
   for (int s = 0; s < k; s++)
   {
      int count = 0;
      for (auto it : v)
      {
         if (it.at(s) == '1')
         {
            count++;
         }
      }
      if (count % 2 == 0)
      {
         cp = cp + '0';
      }
      else
      {
         cp = cp + '1';
      }
   }
   v.push_back(cp);
   display(v);
}
void rowParity(vector<string> v)
{
   vector<string> tdpc;
   for (auto it : v)
   {
      int count = 0;
```

```
        for (int i = 0; i < it.length(); i++)
        {
            if (it.at(i) == '1')
                count++;
        }
        if (count % 2 == 0)
        {
            it = it + '0';
        }
        else
        {
            it = it + '1';
        }
        tdpc.push_back(it);
    }
    columnParity(tdpc);
}
int main()
{
    string s = "";
    int l = 0;
    cout << "enter the binary data bit ";
    cin >> s;
    l = s.length();
    int k;
    cout << "enter the size of each data ";
    cin >> k;
    vector<string> tdp;
    string data = "";
    for (int i = 0; i < l / k; i++)
    {
        for (int j = i * k; j < k + i * k; j++)
        {
            data = data + s.at(j);
        }
        tdp.push_back(data);
        data = "";
    }
    cout << "The data in matrix form : " << endl;
    display(tdp);
    cout << "The generated row and coloumn parity: " << endl;
    rowParity(tdp);
}
```

10. **Convert the dotted decimal into 32 bits IP address and check the class of the IP address using the binary blocks (1 byte or 8 bits length each).**

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
```

```
void display(vector<string> v)
{
   cout << "The ip address in binary : ";

   for (auto it : v)
   {
      cout << it << ".";
   }
   cout << "\b " << endl;
}

string convert(string s)
{
   int n = stoi(s);
   string ans = "";
   int r = 0;
   string c;
   int i = 8;
   while (i > 0 || n > 0)
   {
      r = n % 2;
      c = to_string(r);
      ans = c + ans;
      i--;
      n = n / 2;
   }

   return ans;
}
void findClass(vector<string> v)
{
   string s = v[0];
   string str = "";
   string cls = "";
   for (int i = 0; i < 4; i++)
   {
      str += s[i];
      if (str == "0")
         cls = "A";
      else if (str == "10")
         cls = "B";
      else if (str == "110")
         cls = "C";
      else if (str == "1110")
         cls = "D";
      else if (str == "1111")
         cls = "E";
   }
   cout << "   " << cls;
}
void binaryIp(vector<string> v)
```

```
{
    vector<string> binip;
    for (auto it : v)
    {
        it = convert(it);
        binip.push_back(it);
    }
    display(binip);
    cout << "The class of the ip address : ";
    findClass(binip);
}
int main()
{
    vector<string> v;
    string decip = "";
    cout << "Enter the ip address in dotted decimal : ";
    cin >> decip;
    int l = decip.length();
    string s = "";
    for (int i = 0; i < l; i++)
    {
        if (decip.at(i) == '.')
        {
            v.push_back(s);
            s = "";
        }
        else
        {
            s = s + decip.at(i);
        }
    }
    v.push_back(s);

    binaryIp(v);
}
```