**Aim : : Write a program to add a parity bit to a 7-bit data input by a user and the sender's sites.**

#include <stdio.h>

void main() {

// Declaration of Variables

int binary\_num1, binary\_num2, rem1, rem2, count=0, out=1;

int final[100];

while (out==1){

printf("Enter a binary number with the combination of Os and 1s \n");

scanf("%d", &binary\_num1); // Input the binary number (Os and Is)

printf("Enter the binary number to add \n");

scanf("%d", &binary\_num2); // Input the binary number (Os and Is)

while (binary\_num1 > 0){

rem1= binary\_num1 % 10;

rem2= binary\_num2 % 10;

if (rem1 == 1 && rem2== 1)

{

final[count] = 0;

}

else if(rem1==1 && rem2 ==0)

{

final[count] = 1;

}

else if (rem1 == 0 && rem2==1)

{

final[count] = 1;

}

else{

final[count] = 0;

}

binary\_num1 = binary\_num1/10;

binary\_num2 = binary\_num2/10;

count++;

}

printf("Current Data: ");

for (int i = 0; i < count; ++i) {

printf("%d", final[i]);

}

printf("\nTo add more binary data enter 1, else enter 0 \n");

scanf("%d", &out);

}

/\* Printing the final data \*/

printf ("Final Data: ");

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

{

printf("%d", final[i]);

}

}

**Aim:** Write a program to add redundant bit using Hamming code at the sender's side.

#include <math.h>

#include <stdio.h>

// Store input bits

int input[32];

// Store hamming code

int code[32];

int ham\_calc(int, int);

void solve(int input[], int);

// Function to calculate bit for

//ith position

int ham\_calc(int position, int c\_l)

{

int count = 0, i, j;

i = position - 1;

// Traverse to store Hamming Code

while (i <c\_l) {

for(j=i;j<i+position; j++) {

// If current boit is 1

if (code[j] == 1)

count++;

}

// Update i

i = i +2 \*position;

}

if (count%2==0)

return 0;

else

return 1;

}

// Function to calculate hamming code

void solve(int input[], int n)

{

int i, p\_n=0 , c\_l,j, k;

i = 0;

// Find msg bits having set bit

// at x'th position of number

while (n> (int)pow(2, i) - (i + 1)) {

p\_n++;

i++;

}

c\_l=p\_n+n;

j=k=0;

// Traverse the msgBits

for (i=0; i<c\_l; i++) {

// Update the code

if (i == ((int)pow(2, k) - 1)) {

code[i]= 0;

k++;}

// Update the code[i] to the

// input character at index j

else {

code[i] = input[j];

j++;

}}

// Traverse and update the

// hamming code

for (i=0; i<p\_n; i++) {

// Find current position

int position = (int)pow(2, i);

// Find value at current position

int value = ham\_calc(position, c\_l);

// Update the code

code[position - 1] = value;

}

// Print the Hamming Code

printf("\nThe generated Code Word is: ");

for (i=0; i<c\_l; i++)

{

printf("%d", code[i]);

}

}

// Driver Code

void main()

{

int temp\_input, rem, count=0;

printf ("Please enter the 7 Digit Binary number: ");

scanf ("%d", &temp\_input);

while (temp\_input > 0){

rem = temp\_input%10;

input[count] = rem;

count++;

temp\_input = temp\_input/10;

}

int N = 7;

// Function Call

solve(input, N);

}

**Aim: Write a program to add redundant bit using Hamming code at the receiver's side.**

#include <math.h>

#include <stdio.h>

int d[12];

int pl,p2,p4,p8;

void main()

{

printf("Enter 11 bits of data one by one\n");

scanf("%d",&d[1]);

scanf("%d",&d[2]);

scanf("%d",&d[3]);

scanf("%d",&d[4]);

scanf("%d",&d[5]);

scanf("%d",&d[6]);

scanf("%d",&d[7]);

scanf("%d",&d[8]);

scanf("%d",&d[9]);

scanf("%d",&d[10]);

scanf("%d",&d[11]);

int c=0;

for(int i=11;i>0;i--)

{

printf("%d",d[i]);

}

d[1]=d[1]^d[3]^d[5]^d[7]^d[9]^d[11];

d[2]=d[2]^d[3]^d[6]^d[7]^d[10]^d[11];

d[4]=d[4]^d[5]^d[6]^d[7];

d[8]=d[8]^d[10]^d[11];

//p1=d[1];

//p2=d[2];

//p4= d[4];

//p8=d[8];

for (int i=1;i<=11;i++)

{

if(i==pow(2,c))

{

if(d[i]==1)

{

d[i]=1;

}

else

{

d[i]=0;

}

c++;

}

}

//if(p1==)

printf(“\n The corrected Hamming Code is :”);

for(int i=11;i>0;i--)

{

printf("%d",d[i]);

}

}

**Aim: Write a program to determine the class of given IPV4 address in dotted decimal or binary notation.**

#include <math.h>

#include <string.h>

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[cnt++]=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("\nlp 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 Adress\n");

return 0;

}

**Aim: To Implement Checksum Algorithm using C++**

// C++ implementation of the above approach

#include <bits/stdc++.h>

using namespace std;

// Function to find the One's complement

// of the given binary string

string Ones\_complement(string data)

{

for (int i = 0; i < data.length(); i++) {

if (data[i] == '0')

data[i] = '1';

else

data[i] = '0';

}

return data;

}

// Function to return the checksum value of

// the given string when divided in K size blocks

string checkSum(string data, int block\_size)

{

// Check data size is divisible by block\_size

// Otherwise add '0' front of the data

int n = data.length();

if (n % block\_size != 0) {

int pad\_size = block\_size - (n % block\_size);

for (int i = 0; i < pad\_size; i++) {

data = '0' + data;

}

}

// Binary addition of all blocks with carry

string result = "";

// First block of data stored in result variable

for (int i = 0; i < block\_size; i++) {

result += data[i];

}

// Loop to calculate the block

// wise addition of data

for (int i = block\_size; i < n; i += block\_size) {

// Stores the data of the next block

string next\_block = "";

for (int j = i; j < i + block\_size; j++) {

next\_block += data[j];

}

// Stores the binary addition of two blocks

string additions = "";

int sum = 0, carry = 0;

// Loop to calculate the binary addition of

// the current two blocls of k size

for (int k = block\_size - 1; k >= 0; k--) {

sum += (next\_block[k] - '0')

+ (result[k] - '0');

carry = sum / 2;

if (sum == 0) {

additions = '0' + additions;

sum = carry;

}

else if (sum == 1) {

additions = '1' + additions;

sum = carry;

}

else if (sum == 2) {

additions = '0' + additions;

sum = carry;

}

else {

additions = '1' + additions;

sum = carry;

}

}

// After binary add of two blocks with carry,

// if carry is 1 then apply binary addition

string final = "";

if (carry == 1) {

for (int l = additions.length() - 1; l >= 0;

l--) {

if (carry == 0) {

final = additions[l] + final;

}

else if (((additions[l] - '0') + carry) % 2

== 0) {

final = "0" + final;

carry = 1;

}

else {

final = "1" + final;

carry = 0;

}

}

result = final;

}

else {

result = additions;

}

}

// Return One's complements of result value

// which represents the required checksum value

return Ones\_complement(result);}

// Function to check if the received message

// is same as the senders message

bool checker(string sent\_message,

string rec\_message,

int block\_size)

{

// Checksum Value of the senders message

string sender\_checksum

= checkSum(sent\_message, block\_size);

// Checksum value for the receivers message

string receiver\_checksum = checkSum(

rec\_message + sender\_checksum, block\_size);

// If receivers checksum value is 0

if (count(receiver\_checksum.begin(),

receiver\_checksum.end(), '0')

== block\_size) {

return true;

}

else {

return false;

}

}

// Driver Code

int main()

{

string sent\_message

= "10000101011000111001010011101101";

string recv\_message

= "10000101011000111001010011101101";

int block\_size = 8;

if (checker(sent\_message,

recv\_message,

block\_size)) {

cout << "No Error";

}

else {

cout << "Error";

}

return 0;

}

**Aim: To Implement Pure and Slotted Aloha using C++**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

using namespace std;

class AlohaSystem {

private:

vector<bool> channel; // Represents the communication channel

int numStations; // Number of stations in the system

public:

AlohaSystem(int n) : numStations(n) {

channel.resize(numStations, false);

srand(time(0));

}

void pureAloha() {

for (int i = 0; i < numStations; ++i) {

if (channel[i]) {

cout << "Station " << i + 1 << ": Collision detected, retrying...\n";

} else {

if (rand() % 2 == 0) {

cout << "Station " << i + 1 << ": Frame transmitted successfully!\n";

channel[i] = true;

} else {

cout << "Station " << i + 1 << ": Transmission failed, retrying...\n";

}

}

}

}

void slottedAloha() {

for (int i = 0; i < numStations; ++i) {

if (channel[i]) {

cout << "Station " << i + 1 << ": Collision detected, retrying...\n";

} else {

if (rand() % 2 == 0) {

cout << "Station " << i + 1 << ": Frame transmitted successfully!\n";

channel[i] = true;

} else {

cout << "Station " << i + 1 << ": Transmission failed, retrying...\n";

}

}

}

}

};

int main() {

int numStations;

cout << "Enter the number of stations: ";

cin >> numStations;

AlohaSystem aloha(numStations);

cout << "Pure Aloha:\n";

aloha.pureAloha();

cout << "\nSlotted Aloha:\n";

aloha.slottedAloha();

return 0;

}

**Aim:** **To implement Subnetting Mask.**

#include <stdio.h>

// Function to perform subnetting

void subnetting(char ip[], int subnetBits) {

// Convert IP address to binary

unsigned long ipAddress = 0;

sscanf(ip, "%lu", &ipAddress);

// Create subnet mask

unsigned long subnetMask = (~0UL) << (32 - subnetBits);

// Calculate network address

unsigned long networkAddress = ipAddress & subnetMask;

// Display results

printf("IP Address: %s\n", ip);

printf("Subnet Mask: %lu\n", subnetMask);

printf("Network Address: %lu\n", networkAddress);

}

int main() {

char ip[16];

int subnetBits;

// Get IP address and subnet bits from the user

printf("Enter IP address (in decimal format): ");

scanf("%s", ip);

printf("Enter Subnet Bits: ");

scanf("%d", &subnetBits);

// Perform subnetting

subnetting(ip, subnetBits);

return 0;

}