

Experiment - 7

Objective: To write a program to Calculate Checksum

Theory:

A checksum is a error detection method in Data Communication. It is used for errors which may have been introduced during transmission or storage. It is usually applied to an installation file after it is received from the download server.

Checksum method can only detect errors but is unable to correct the error.

In this method a checksum is calculated based on the given binary strings which is sent with the data as redundant bits. This data + checksum is received at receiver end and checksum is calculated again, if checksum is 0 it means no error in data received, else there exists some error in the received data.

Program:

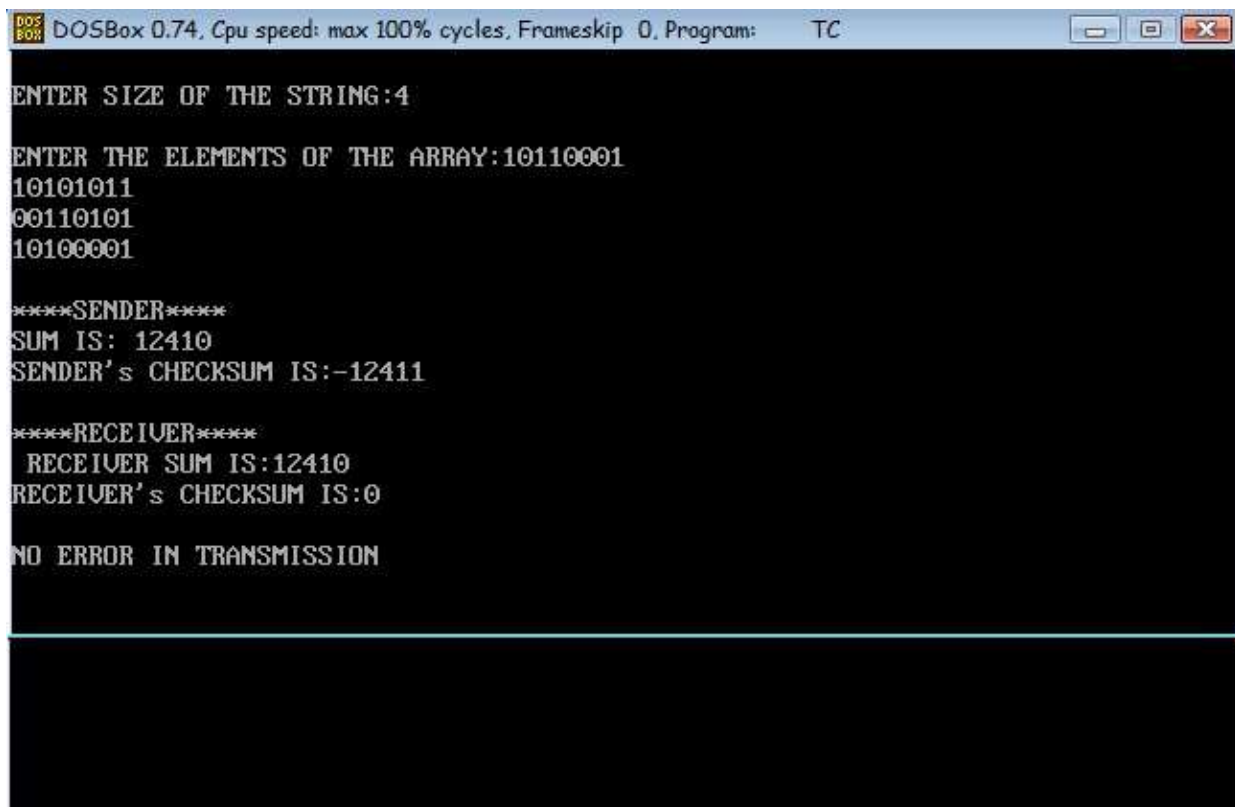
```
1. #include<conio.h>
2. #include<math.h>
3. #include<stdio.h>
4. intsender(int b[10],int k)
5. {
6.     intchecksum,sum=0,i;
7.     printf("\n****SENDER****\n");
8.
9.     for(i=0;i<k;i++)
10.        sum+=b[i];
11.     printf("SUM IS: %d",sum);
12.
13.     checksum=~sum;
14.     printf("\nSENDER's CHECKSUM IS:%d",checksum);
15.     return checksum;
16. }
17.
18. intreceiver(int c[10],intk,intscheck)
19. {
20.     intchecksum,sum=0,i;
21.     printf("\n\n****RECEIVER****\n");
22.     for(i=0;i<k;i++)
23.        sum+=c[i];
24.     printf(" RECEIVER SUM IS:%d",sum);
25.     sum=sum+scheck;
26.     checksum=~sum;
27.     printf("\nRECEIVER's CHECKSUM IS:%d",checksum);
28.     return checksum;
```

```

29. }
30. main()
31. {
32.     int a[10], i, m, scheck, rcheck;
33.     clrscr();
34.     printf("\nENTER SIZE OF THE STRING:");
35.     scanf("%d", &m);
36.     printf("\nENTER THE ELEMENTS OF THE ARRAY:");
37.     for(i=0; i<m; i++)
38.         scanf("%d", &a[i]);
39.     scheck = sender(a, m);
40.     rcheck = receiver(a, m, scheck);
41.     if(rcheck == 0)
42.         printf("\n\nNO ERROR IN TRANSMISSION\n\n");
43.     else
44.         printf("\n\nERROR DETECTED"); [
45.         getch();
46.     }

```

Output:



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
ENTER SIZE OF THE STRING:4
ENTER THE ELEMENTS OF THE ARRAY:10110001
10101011
00110101
10100001

****SENDER****
SUM IS: 12410
SENDER'S CHECKSUM IS:-12411

****RECEIVER****
RECEIVER SUM IS:12410
RECEIVER'S CHECKSUM IS:0

NO ERROR IN TRANSMISSION

```

Experiment - 8

Objective: Write a Program to compute Hamming Distance between Two Data word

Theory:

You are given two strings of equal length, you have to find the Hamming Distance between these string. Where the Hamming distance between two strings of equal length is the number of positions at which the corresponding character are different.

Program:

```
void main()
{ int a[10];
  int b[10];
  int c=0; int
  i,n; clrscr();
  printf("enter the length of data code\n"); scanf("%d",&n);
  printf("enter the sender side data code\n"); for(i=0;i<n;i++)
  {
    scanf("%d",&a[i]);
  }
  printf("enter the reciver side datacode\n"); for(i=0;i<n;i++)
  {
    scanf("%d",&b[i]);
  }
  for(i=0;i<n;i
  ++){
    if(a[i]!=b[i])
    { c++; }
  }
  printf("hamming distance is
  %d",c); getch();

}
```

OUTPUT:

enter the length of data code 6 enter the
sender side data code

1

0

0

1

1 1 enter the receiver side data code

1

1

1

1

1

1

Hamming distance is 2

Experiment -9

Objective: Write a program for error detecting code using CRC-CCITT (16-bits).

Theory- It does error checking via polynomial division. In general, a bit string $b_{n-1}b_{n-2}b_{n-3}\dots b_2b_1b_0$ As $b_{n-1}X^{n-1} + b_{n-2}X^{n-2} + b_{n-3}X^{n-3} + \dots + b_2X^2 + b_1X^1 + b_0$

For example- 10010101110 can be expressed as $X^{10} + X^7 + X^5 + X^3 + X^2 + X^1$

All computations are done in modulo 2.

Algorithm:-

1. Given a bit string, append 0s to the end of it (the number of 0s is the same as the degree of the generator polynomial) let $B(x)$ be the polynomial corresponding to B.
2. Divide $B(x)$ by some agreed on polynomial $G(x)$ (generator polynomial) and determine the remainder $R(x)$. This division is to be done using Modulo 2 Division.
3. Define $T(x) = B(x) - R(x)$
($T(x)/G(x) \Rightarrow$ remainder 0)
4. Transmit T, the bit string corresponding to $T(x)$.
5. Let T' represent the bit stream the receiver gets and $T'(x)$ the associated polynomial. The receiver divides $T'(x)$ by $G(x)$. If there is a 0 remainder, the receiver concludes $T = T'$ and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission.

Program :

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#define N strlen(g)
char t[128], cs[128], g[]="100010000000100001";
int a, e, c;
void xor()
{
    for(c=1;c<N;c++)
        cs[c]=((cs[c]==g[c])?'0':'1'); }
void crc()
{
    for(e=0;e<N;e++) cs[e]=t[e];
    do {
        if(cs[0]=='1') xor();
        for(c=0;c<N-1;c++) cs[c]=cs[c+1];
```

```

    cs[c]=t[e++];
}
while(e<=a+N-1);
}
void main(){
clrscr();
printf("\nEnter poly : ");
scanf("%s",t);
printf("\nGenerating Polynomial is :
%s",g);
a=strlen(t);
for(e=a;e<a+N-1;e++)
t[e]='0';
printf("\nModified t[u] is : %s",t);
crc();
printf("\nChecksum is : %s",cs);
for(e=a;e<a+N-1;e++)
t[e]=cs[e-a];
printf("\nFinal Codeword is : %s",t);
scanf("%d",&e); if(e==0) {
printf("Enter position where error is to inserted : ");
scanf("%d",&e);
t[e]=(t[e]=='0')?'1':'0';
printf("Erroneous data : ");
crc();
for(e=0;(e<N-1)&&(cs[e]!='1');e++);
if(e<N1)
printf("Error detected.");
else
printf("No Error Detected.");
getch();
}

```

OUTPUT :

Enter poly : 1011101

Generating Polynomial is : 10001000000100001

Modified t[u] is : 1011101000000000000000

Checksum is : 1000101101011000

Final Codeword is : 10111011000101101011000

Test Error detection 0(yes) 1(no) ? : 0

Enter position where you want to insert error : 3

Erroneous data : 101**0**1011000101101011000

Error detected.

Enter poly : 1011101

Generating Polynomial is : 10001000000100001

Modified t[u] is : 1011101000000000000000

Checksum is : 1000101101011000

Final Codeword is : 10111011000101101011000 Test

Error detection 0(yes) 1(no) ? : 1 No Error Detected.

Experiment - 10

Objective: To write a program to Implement Hamming Code

Theory:

Hamming code is a popular error detection and error correction method in data communication. Hamming code can only detect 2 bit error and correct a single bit error which means it is unable to correct burst errors if may occur while transmission of data.

Program:

```
#include<stdio.>
voidmain(){
intdata[10];
intdataatrec[10],c,c1,c2,c3,i;
printf("Enter 4 bits of data one by
one\n");
scanf("%d",&data[0]);
scanf("%d",&data[1]);
scanf("%d",&data[2]);
scanf("%d",&data[4]);

//Calculation of even parity

data[6]=data[0]^data[2]^data[4];
data[5]=data[0]^data[1]^data[4];
data[3]=data[0]^data[1]^data[2];
printf("\nEncoded data is\n");
for(i=0;i<7;i++)
printf("%d",data[i]);
printf("\n\nEnter received data bits one by one\n");
for(i=0;i<7;i++)
scanf("%d",&dataatrec[i]);
c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];
c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];
c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];
c=c3*4+c2*2+c1;

if(c==0){
printf("\nNo error while transmission of data\n");
}
else{
printf("\nError on position %d",c);
printf("\nData sent : ");
for(i=0;i<7;i++)
printf("%d",data[i]);
printf("\nData received : ");
for(i=0;i<7;i++)
printf("%d",dataatrec[i]);
printf("\nCorrect message is\n");
```



```
//if erroneous bit is 0 we complement it else vice versa
if(dataatrec[7-c]==0)
dataatrec[7-c]=1;
else
dataatrec[7-c]=0;
for(i=0;i<7;i++){
printf("%d",dataatrec[i]);
}
}
}
```

Output:

Enter 4 bits of data one by one

1

0

1

0

Encoded data is

1

0

1

0

0

1

0

E

n

t

e

r

received data bits one by one

1

0

1

0

0

1

0

No error while transmission of data

EXPERIMENT – 11

Objective: To write a Program to Compute Link Utilization on Stop and Wait Protocol.

Theory:

Stop-and-Wait Protocol: In stop-and-wait protocol, the sender transmits a single frame and waits for acknowledgment from the receiver before sending the next frame. If acknowledgment is not received within a timeout period, the sender retransmits the frame. This ensures reliable data transfer but can lead to inefficiency due to idle time.

SOURCE CODE:

a) Stop and Wait protocol:-

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

#define MESSAGE_SIZE 100

void sender(char* message, char*
ack) {    printf("Sending message -
%s\n", message);    sleep(1);
    printf("Waiting for
acknowledgment...\n");    strcpy(ack,
"ACK");
    printf("Acknowledgment received\n");
}

void receiver(char* ack) {
    printf("Did you receive the message? (Type 'ACK' if
yes): ");    scanf("%s", ack);
}

int main() {
    char message[MESSAGE_SIZE] =
"Hello, World!";    char ack[4];

    sender(message, ack);
    receiver(ack);

    printf("Message sent successfully\n");

    return 0; }
```

OUTPUT:

```
Sender: Sending message - Hello, World!  
Sender: Waiting for acknowledgment...  
Receiver: Did you receive the message? (Type 'ACK' if yes): ACK  
Sender: Acknowledgment received  
Sender: Message sent successfully
```

EXPERIMENT – 12

Objective: To write a program for bit stuffing used in HDLC.

Theory:

A Bit Stuffing program in C is a software application written in the C programming language that implements the process of bit-stuffing. It is a technique used in data communication to ensure synchronization and reliable transmission of information.

Bit stuffing is a process used in data communication to break up sequences of bits that might be misinterpreted by a receiver. For example, a certain pattern of bits may signal the end of a frame. If these bits naturally appear in the data, the receiver might falsely interpret it as the end of the frame. To prevent this, an extra bit (usually a '0') is inserted or 'stuffed' after a certain number of consecutive '1' bits in the data stream.

Efficient bit stuffing algorithms are essential for various communication protocols such as HDLC (High-Level Data Link Control) and CAN (Controller Area Network), where data integrity and synchronization are critical. Engineers and developers often employ C for such tasks due to its efficiency, low-level control, and portability across different platforms and hardware architectures. Therefore, mastering bit stuffing in C can be valuable for those working in the field of embedded systems, networking, and telecommunications.

SOURCE CODE:

```
#include <stdio.h>
#include <string.h>
int main() {
    char data[100], stuffedData[200];
    int i, count = 0, j = 0;
    printf("Enter the data: ");
    scanf("%s", data);
    for(i = 0; i < strlen(data); i++) {
        if(data[i] == '1') {
            count++;
            stuffedData[j++] = data[i];
        } else {
            count = 0;
            stuffedData[j++] = data[i];
        }
    }
    if(count == 5) {
        count = 0;
```

```
        stuffedData[j++] = '0';
    }
}
stuffedData[j] = '\0';
printf("Data after bit stuffing: %s\n", stuffedData);
return 0;
}
```

OUTPUT:

```
E:\Tailwind>cd "e:\Tailwind\src\" && gcc p.c -o p && "e:\Tailwind\src\"p
Enter the data: 11011111
Data after bit stuffing: 110111110
```

EXPERIMENT – 13

Objective: To write a program to find Class of IP address and Network Address for Given IP address.

Theory:

Each computer which is connected to the network has an identifier to recognise connected computer address, known as IP (Internet Protocol) Address.

An IPv4 has 4 octets having decimal value between 0 to 255 and depending on the first octet's value, IP Addresses divided into 5 classes:

Class A	1	to	126
Class B	128	to	191
Class C	192	to	223
Class D	224	to	239
Class E	240	to	254

SOURCE CODE:

```
// C program to determine class, Network
// and Host ID of an IPv4 address
#include<stdio.h>
#include<string.h>

// Function to find out the Class
char findClass(char str[])
{
    // storing first octet in arr[] variable
    char arr[4];
    int i = 0;
    while (str[i] != '.')
    {
        arr[i] = str[i];
        i++;
    }
    i--;

    // converting str[] variable into number for
    // comparison
    int ip = 0, j = 1;
    while (i >= 0)
    {
        ip = ip + (str[i] - '0') * j;
        j = j * 10;
        i--;
    }
}
```

```

// Class A
if (ip >= 1 && ip <= 126)
return 'A';

// Class B
else if (ip >= 128 && ip <= 191)
return 'B';

// Class C
else if (ip >= 192 && ip <= 223)
return 'C';

// Class D
else if (ip >= 224 && ip <= 239)
return 'D';

// Class E
else
return 'E';
}

// Function to separate Network ID as well as
// Host ID and print them
void separate(char str[], char ipClass)
{
// Initializing network and host array to NULL
char network[12], host[12];
for (int k = 0; k < 12; k++)
network[k] = host[k] = '\0';

// for class A, only first octet is Network ID
// and rest are Host ID
if (ipClass == 'A')
{
int i = 0, j = 0;
while (str[j] != '.')
network[i++] = str[j++];
i = 0;
j++;
while (str[j] != '\0')
host[i++] = str[j++];
printf("Network ID is %s\n", network);
printf("Host ID is %s\n", host);
}
}

```

```

// for class B, first two octet are Network ID
// and rest are Host ID
else if (ipClass == 'B')
{
int i = 0, j = 0, dotCount = 0;

// storing in network[] up to 2nd dot
// dotCount keeps track of number of
// dots or octets passed
while (dotCount < 2)
{
    network[i++] = str[j++];
    if (str[j] == '.')
        dotCount++;
}
i = 0;
j++;

while (str[j] != '\0')
    host[i++] = str[j++];

printf("Network ID is %s\n", network);
printf("Host ID is %s\n", host);
}

// for class C, first three octet are Network ID
// and rest are Host ID
else if (ipClass == 'C')
{
int i = 0, j = 0, dotCount = 0;

// storing in network[] up to 3rd dot
// dotCount keeps track of number of
// dots or octets passed
while (dotCount < 3)
{
    network[i++] = str[j++];
    if (str[j] == '.')
        dotCount++;
}

i = 0;
j++;

while (str[j] != '\0')
    host[i++] = str[j++];

```



```

printf("Network ID is %s\n", network);
printf("Host ID is %s\n", host);
}

// Class D and E are not divided in Network
// and Host ID
else
printf("In this Class, IP address is not"
" divided into Network and Host ID\n");
}

// Driver function is to test above function
int main()
{
char str[] = "145.160.017.001";
char ipClass = findClass(str);
printf("Given IP address belongs to Class %c\n",ipClass);
separate(str, ipClass);
return 0;
}

```

OUTPUT:

```

e:\Tailwind\src>cd "e:\Tailwind\src\" && gcc p.c -o p && "e:\Tailwind\src\"p
Enter IP Address (xxx.xxx.xxx.xxx format): 145.160.017.001
Given IP address belongs to Class B
Network ID is 145.160

```