Frame Generation

PPP frame format was chosen to closely resemble the frame format of HDLC (High-level Data Link Control), a widely used instance of an earlier family of protocols, since there was no need to reinvent the wheel.

The primary difference between PPP and HDLC is that PPP is byte oriented rather than bit oriented. In particular, PPP uses byte stuffing and all frames are an integral number of bytes. HDLC uses bit stuffing and allows frames of, say, 30.25 bytes.

There is a second major difference in practice, however. HDLC provides reliable transmission with a sliding window, acknowledgements, and timeouts in the manner we have studied. PPP can also provide reliable transmission in noisy environments, such as wireless networks; the exact details are defined in RFC 1663. However, this is rarely done in practice.

Code for Frame Generation

```
#include<iostream>
#include<cstdio>
#define MAX_SIZE 100
#define Generator_Size 16
#define Gen_Size Generator_Size+1
using namespace std;
void Scan_Data(bool *Source,int &Source_Size)
{
   // for scanning the Frame and Generator into respective variables
   char c=getchar();
        Source_Size=0;
        while(c!='\n')
   {
      Source[Source_Size++]=c=='0'?0:1;
      c=getchar();
   }
}
struct Frame_Structure
{
        bool Flag_Sequence[8]={0,1,1,1,1,1,1,0};
        bool Source_Add[MAX_SIZE],Destination_Add[MAX_SIZE];
        bool Data[MAX_SIZE];
   bool Check_Sum[Generator_Size];
   int Source_Size,Destination_Size,Data_Size;
};
void Display(bool *Source,int Source_Size)
{
   for(int i=0;i<Source_Size;i++)
      cout<<Source[i];
```

```
}
void Print(Frame_Structure& x)
{
   cout<<"\nFlag Sequence\tSource Address\tDestination Address\t\t\tData\t\t\tCheck
Sum\tFlag Sequence\n";
        cout<<"_____\t_____\t_____
_____ _____ _____\n";
        cout<<"   ";
        Display(x.Flag_Sequence,8);
        cout<<"\t   ";
        Display(x.Source_Add,x.Source_Size);
        cout<<"\t  ";
        Display(x.Destination_Add,x.Destination_Size);
        cout<<"\t\t";
        Display(x.Data,x.Data_Size);
        cout<<"     ";
        Display(x.Check_Sum,Generator_Size);
        cout<<"   ";
        Display(x.Flag_Sequence,8);
        cout<<endl;
}
void Scan_Address(Frame_Structure& x)
{
        cout<<"\nEnter Source Address : ";
        Scan_Data(x.Source_Add,x.Source_Size);
        cout<<"\nEnter Destination Address : ";
        Scan_Data(x.Destination_Add,x.Destination_Size);
}
void Calc_Check_Sum(Frame_Structure& x)
{
   int i,j,flag;
   bool Generator[Gen_Size]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1}; // CRC-16
x^16+x^12+x^5+x^0
   bool Hold[Gen_Size],Frame[x.Data_Size+Generator_Size];
   for(i=0;i<x.Data_Size;i++)
        Frame[i]=x.Data[i];
   int Frame_Size=x.Data_Size;
   for(i=0;i<Generator_Size;i++)
        Frame[Frame_Size++]=0;
   i=0;
   while(!Frame[i]) // when Hold < Generator
      i++;
   for(;i<Gen_Size;i++)   // load Hold with initial data
      Hold[i]=Frame[i];
   flag=0;
   for(j=Gen_Size;j<Frame_Size;)
   {
      int k;
```

```
        for(i=flag,k=1;k<Gen_Size;i++,k++)  // XORing Generator with dividend(i.e., Hold)
            Hold[i%Gen_Size]=Hold[(i+1)%Gen_Size]^Generator[k];
        Hold[i%Gen_Size]=Frame[j];
        j++;
        for(;Hold[flag%Gen_Size]!=Generator[0] && j<Frame_Size;j++,flag++) // when Hold
< Generator
            Hold[flag%Gen_Size]=Frame[j];
    }
    for(i=0,j=flag+1;i<Generator_Size;j++,i++) // XORing remainder with dividend(i.e.,
Frame)
        x.Check_Sum[i]=Hold[j%Gen_Size];
}
int main()
{
          Frame_Structure var;
          cout<<"\nEnter Data : ";
          Scan_Data(var.Data,var.Data_Size);
          Scan_Address(var);
          Calc_Check_Sum(var);
          Print(var);
}
```

Results : -