CRC 12 :

```
#include <stdio.h>
#include <string.h>
int main() {
    char data[100], poly[20];
    int i, j, data_len, poly_len;
    printf("Enter the binary data frame: ");
    scanf("%s", data);
    printf("Enter the CRC-12 generation polynomial (13 bits):
");
    scanf("%s", poly);
    data_len = strlen(data);
    poly_len = strlen(poly);
    char frame[120];
    strcpy(frame, data);
    for(i = 0; i < poly_len - 1; i++)
        frame[data_len + i] = '0';
    frame[data_len + i] = '\0';
    for(i = 0; i < data_len; i++) {
        if(frame[i] == '1') {
            for(j = 0; j < poly_len; j++)
                frame[i + j] = ((frame[i + j] - '0') ^ (poly[j] - '0')) +
'0';
        }
    }
    char crc[20];
    for(i = 0; i < poly_len - 1; i++)   crc[i] = frame[data_len +
i];
    crc[i] = '\0';
    printf("\nComputed CRC: %s\n", crc);
    printf("Transmitted Frame (Data + CRC): %s%s\n", data,
crc);
}
```

CRC 16 :

```
#include <stdio.h>
#include <string.h>
int main() {
    char data[100], poly[20];
    int i, j, data_len, poly_len;
    printf("Enter the binary data frame: ");
    scanf("%s", data);
    printf("Enter the CRC-16 generation polynomial (17 bits):
");
    scanf("%s", poly);
    data_len = strlen(data);
    poly_len = strlen(poly);
    char frame[120]; // make sure enough size
    strcpy(frame, data);
    for(i = 0; i < poly_len - 1; i++)
        frame[data_len + i] = '0';
    frame[data_len + i] = '\0';
    for(i = 0; i < data_len; i++) {
        if(frame[i] == '1') {
            for(j = 0; j < poly_len; j++)
                frame[i + j] = ((frame[i + j] - '0') ^ (poly[j] - '0')) +
'0';
        }
    }
    char crc[20];
    for(i = 0; i < poly_len - 1; i++)
        crc[i] = frame[data_len + i];
    crc[i] = '\0';
    printf("\nComputed CRC: %s\n", crc);
    printf("Transmitted Frame (Data + CRC): %s%s\n", data,
crc);
}
```

CRC CCIT:

```
#include <stdio.h>
#define POLY 0x1021
#define INIT_CRC 0xFFFF
unsigned short crc_ccitt(unsigned char *data, int len) {
    unsigned short crc = INIT_CRC;
    for (int i = 0; i < len; i++) {
        crc ^= (data[i] << 8);
        for (int j = 0; j < 8; j++) {
            if (crc & 0x8000)
                crc = (crc << 1) ^ POLY;
            else
                crc <<= 1;
        }
    }
    return crc;
}
int main() {
    unsigned char data = 0x31;  // example data (ASCII for '1')
    unsigned short crc = crc_ccitt(&data, 1);
    printf("CRC after 8 bit iteration: 0x%X\n", crc);
    return 0;
}
```

WEEK – 4: DIJKSTRA'S

```
#include <stdio.h>
#define INF 3000
int main() {
    int cost[20][20], dist[20], path[20], s[20];
    int i, j, u, v, num, n, min;

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter the adjacency matrix (-1 for no edge):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
            if (i == j)
                cost[i][j] = 0;
            else if (cost[i][j] == -1)
                cost[i][j] = INF;
        }
    }

    printf("Enter source vertex (0 to %d): ", n - 1);
    scanf("%d", &v);

    for (i = 0; i < n; i++) {
        s[i] = 0;
        dist[i] = cost[v][i];
        if (dist[i] != INF && i != v)
            path[i] = v;
        else
            path[i] = -1;
    }

    s[v] = 1;
    dist[v] = 0;

    for (num = 1; num < n; num++) {
        min = INF;
        u = -1;
        for (i = 0; i < n; i++) {
            if (!s[i] && dist[i] < min) {
                min = dist[i];
                u = i;
            }
        }

        if (u == -1)
            break;

        s[u] = 1;
        for (j = 0; j < n; j++) {
            if (!s[j] && dist[u] + cost[u][j] < dist[j]) {
                dist[j] = dist[u] + cost[u][j];
                path[j] = u;
            }
        }
    }

    printf("\nVertex\tDistance\tPath\n");
    for (i = 0; i < n; i++) {
        if (dist[i] == INF) {
            printf("%d\tINF\t\tNo path\n", i);
        } else {
            printf("%d\t%d\t\t", i, dist[i]);
            j = i;
            printf("%d", j);
            while (j != v) {
                j = path[j];
                printf("<-%d", j);
            }
            printf("\n");
        }
    }

    return 0;
}
```

ADDITIONAL:

1)Discuss the subnetting for IPv4 addressing.

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

uint32_t ipstr_to_int(const char *s){
    unsigned a,b,c,d;
    sscanf(s,"%u.%u.%u.%u",&a,&b,&c,&d);
    return (a<<24)|(b<<16)|(c<<8)|d;
}

void int_to_ipstr(uint32_t x, char *s){
    sprintf(s, "%u.%u.%u.%u", (x>>24)&0xFF,
(x>>16)&0xFF, (x>>8)&0xFF, x&0xFF);
}

int main(){
    char ipstr[32]; int prefix;
    printf("Enter IPv4 address (a.b.c.d): "); scanf("%s", ipstr);
    printf("Enter prefix length (e.g. 24): "); scanf("%d",
&prefix);
    uint32_t ip = ipstr_to_int(ipstr);
    uint32_t mask = prefix==0 ? 0 : (0xFFFFFFFF << (32-
prefix));
    uint32_t network = ip & mask;
    uint32_t broadcast = network | (~mask);
    char netstr[32], bcaststr[32];
    int_to_ipstr(network, netstr); int_to_ipstr(broadcast,
bcaststr);
    printf("Network: %s\nBroadcast: %s\nSubnet mask:
%u.%u.%u.%u\n", netstr, bcaststr,

(mask>>24)&0xFF,(mask>>16)&0xFF,(mask>>8)&0xFF,mas
k&0xFF);
    return 0;
}
```

2) Implement data packets by using wire shark.

```
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>

int main(){
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_if_t *alldevs;
    if(pcap_findalldevs(&alldevs, errbuf) == -1){ printf("Error:
%s\n", errbuf); return 1;}
```

```
    pcap_t *handle = pcap_open_live(alldevs->name, 65536, 1, 1000, errbuf);
    if(!handle){ printf("Open live failed: %s\n", errbuf); return 1;}
    printf("Capturing on %s ... Ctrl+C to stop\n", alldevs->name);
    struct pcap_pkthdr *header;
    const u_char *pkt;
    for(int i=0;i<10 && pcap_next_ex(handle, &header, &pkt)>=0; ++i){
        printf("Packet %d: len=%d\n", i+1, header->len);
    }
    pcap_close(handle);
    pcap_freealldevs(alldevs);
    return 0;
}
```

3) Implement HTTP client server experiment.

```
/* server.c */
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main(){
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr={0};
    addr.sin_family=AF_INET; addr.sin_port=htons(8080); addr.sin_addr.s_addr=INADDR_ANY;
    bind(sock, (struct sockaddr*)&addr, sizeof(addr));
    listen(sock, 5);
    printf("HTTP server on port 8080\n");
    int c = accept(sock, NULL, NULL);
    char buf[1024];
    read(c, buf, sizeof(buf)-1);
    printf("Request:\n%s\n", buf);
    const char *resp = "HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\nContent-Length: 12\r\n\r\nHello World\n";
    write(c, resp, strlen(resp));
    close(c); close(sock);
    return 0;
}
/* client.c */
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main(){
    int s = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in srv={0}; srv.sin_family=AF_INET; srv.sin_port=htons(8080);
    inet_pton(AF_INET, "127.0.0.1", &srv.sin_addr);
    connect(s, (struct sockaddr*)&srv, sizeof(srv));
    char *req = "GET / HTTP/1.1\r\nHost: localhost\r\n\r\n";
    send(s, req, strlen(req), 0);
    char buf[2048]; int r = recv(s, buf, sizeof(buf)-1, 0);
    buf[r]=0; printf("Response:\n%s\n", buf);
    close(s);
    return 0;
}
```

4) Perform X-Or operation for 8-bit data for checksum

```
#include <stdio.h>

int main(){
    int n; printf("Number of bytes: "); scanf("%d",&n);
    unsigned int x, checksum=0;
    printf("Enter bytes in hex (e.g. 3f) or decimal:\n");
    for(int i=0;i<n;++i){ scanf("%x",&x); checksum ^= x & 0xFF; }
    printf("XOR checksum (8-bit)= %02x\n", checksum & 0xFF);
    return 0;
}
```