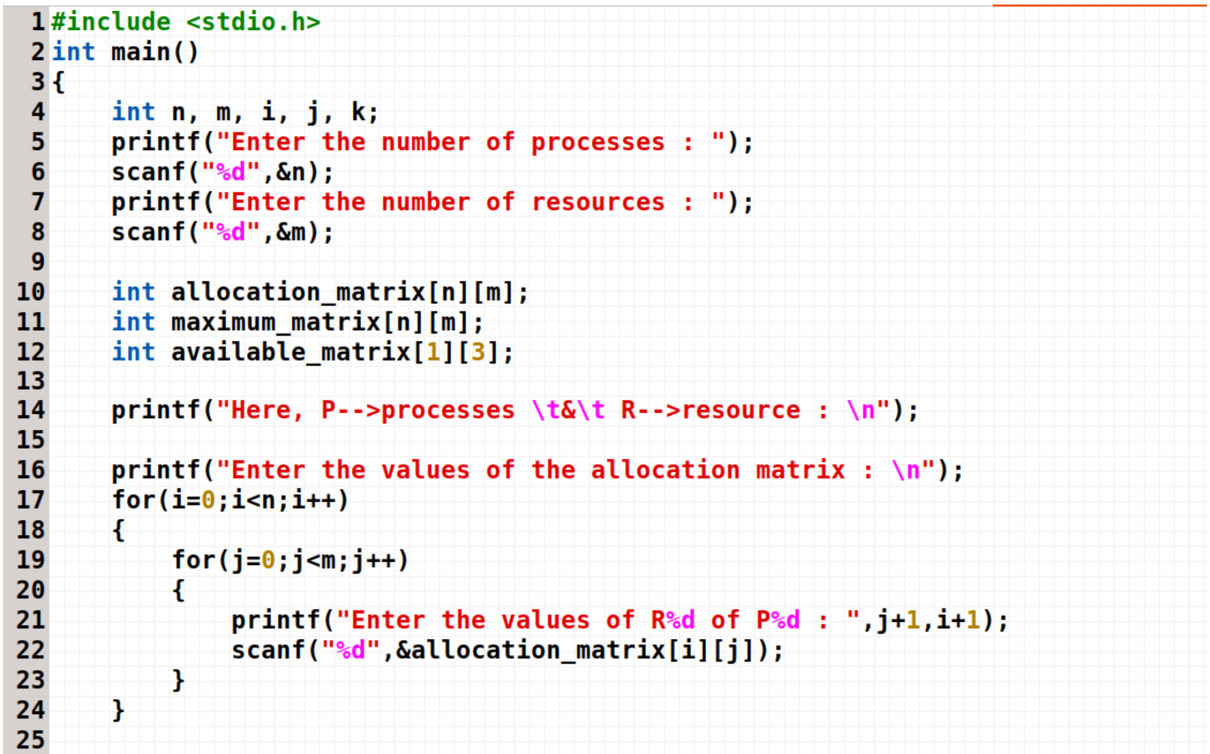
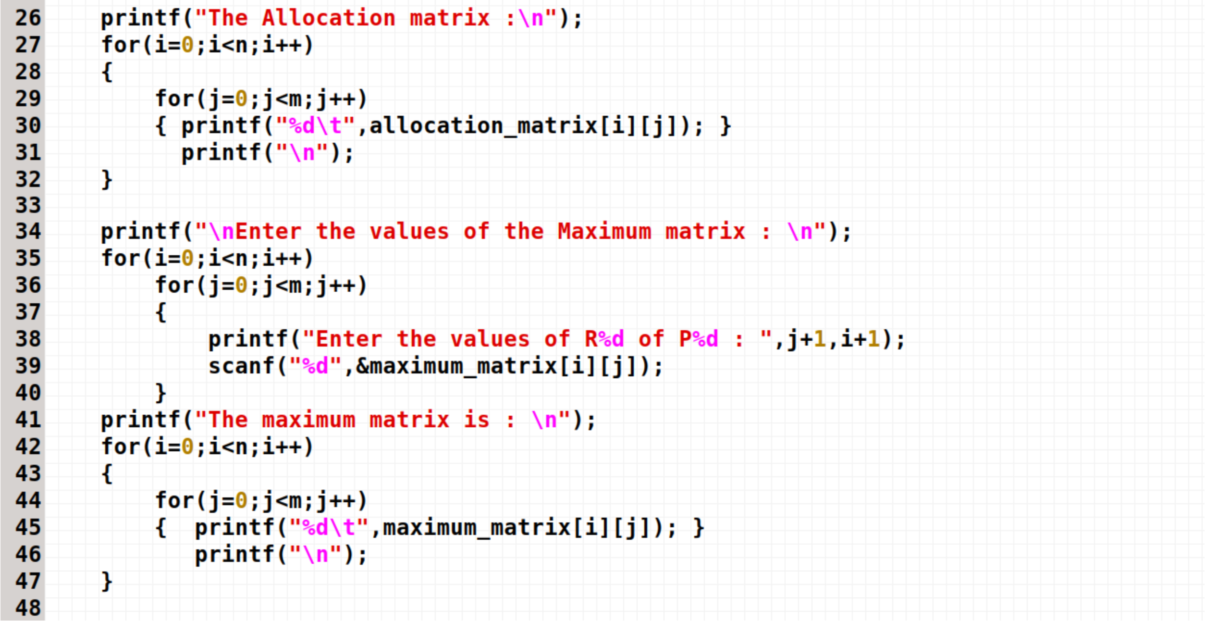
Assignment-2 Prashanth.S(19MID0020)

Banker’s Algorithm

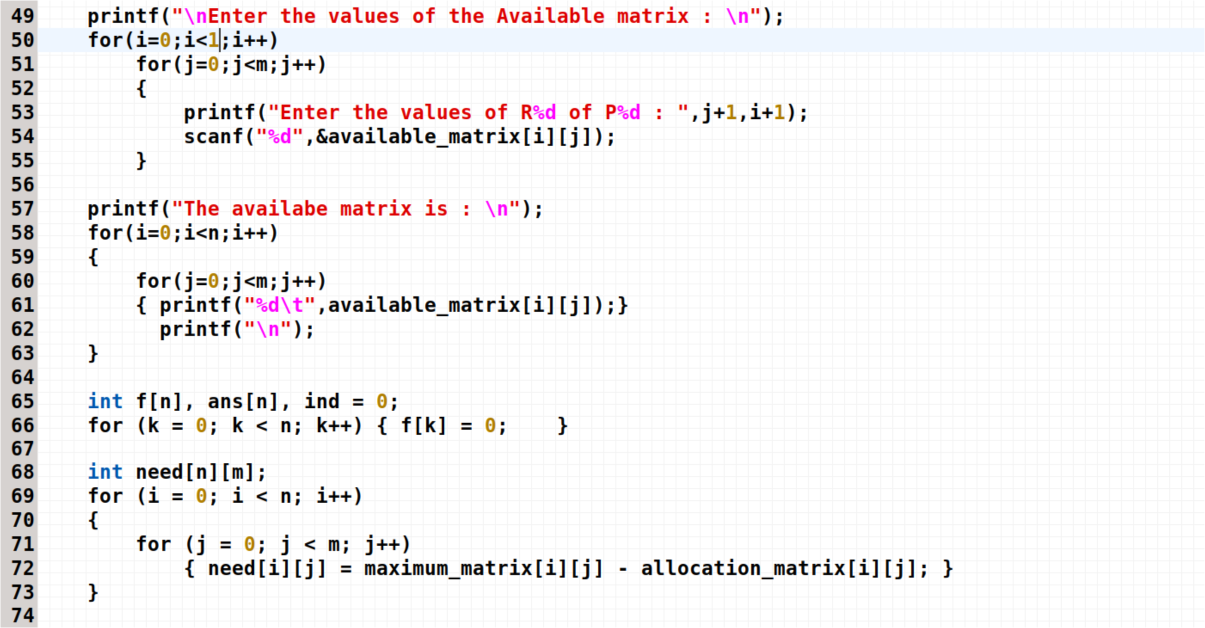
Code:



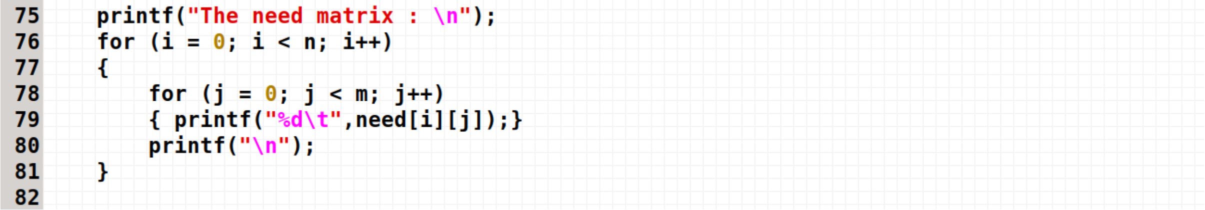
```c
#include <stdio.h>
int main()
{
    int n, m, i, j, k;
    printf("Enter the number of processes : ");
    scanf("%d",&n);
    printf("Enter the number of resources : ");
    scanf("%d",&m);

    int allocation_matrix[n][m];
    int maximum_matrix[n][m];
    int available_matrix[1][3];

    printf("Here, P-->processes \t&\t R-->resource : \n");

    printf("Enter the values of the allocation matrix : \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            printf("Enter the values of R%d of P%d : ",j+1,i+1);
            scanf("%d",&allocation_matrix[i][j]);
        }
    }

```



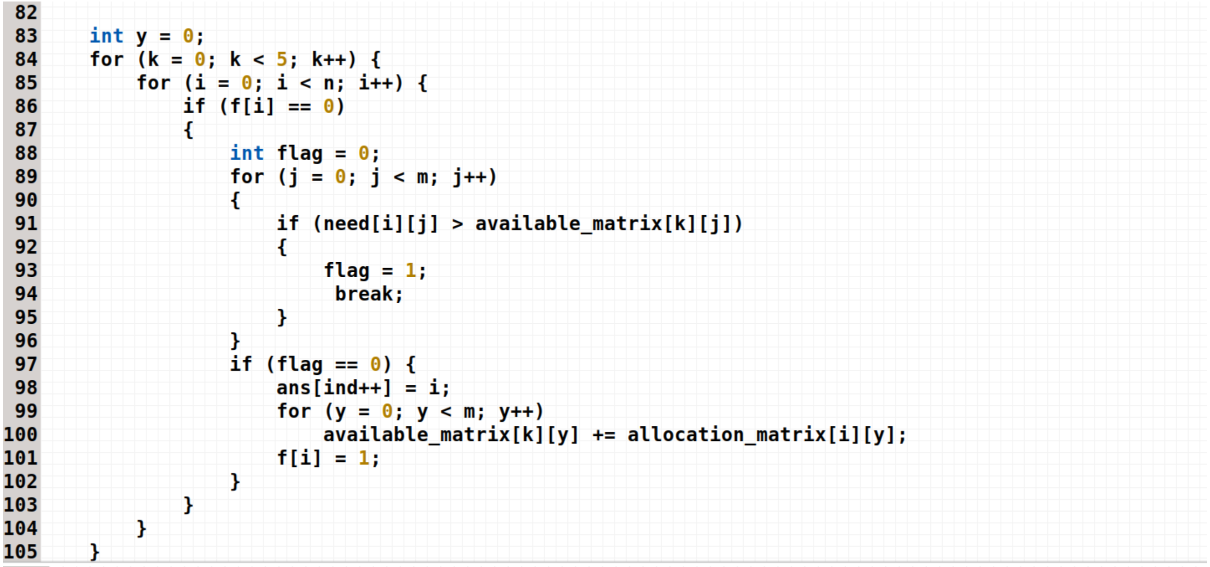
```c
    printf("The Allocation matrix :\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        { printf("%d\t",allocation_matrix[i][j]); }
          printf("\n");
    }

    printf("\nEnter the values of the Maximum matrix : \n");
    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
        {
            printf("Enter the values of R%d of P%d : ",j+1,i+1);
            scanf("%d",&maximum_matrix[i][j]);
        }
    printf("The maximum matrix is : \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {  printf("%d\t",maximum_matrix[i][j]); }
           printf("\n");
    }

```



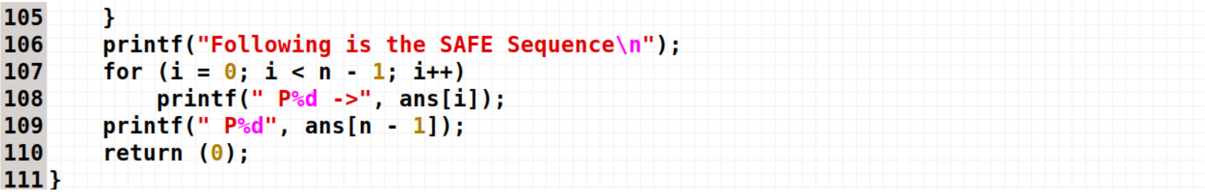
```
    printf("\nEnter the values of the Available matrix : \n");
    for(i=0;i<1;i++)
        for(j=0;j<m;j++)
        {
            printf("Enter the values of R%d of P%d : ",j+1,i+1);
            scanf("%d",&available_matrix[i][j]);
        }

    printf("The availabe matrix is : \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        { printf("%d\t",available_matrix[i][j]);}
          printf("\n");
    }

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) { f[k] = 0;     }

    int need[n][m];
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
            { need[i][j] = maximum_matrix[i][j] - allocation_matrix[i][j]; }
    }

```



```
    printf("The need matrix : \n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        { printf("%d\t",need[i][j]);}
        printf("\n");
    }

```

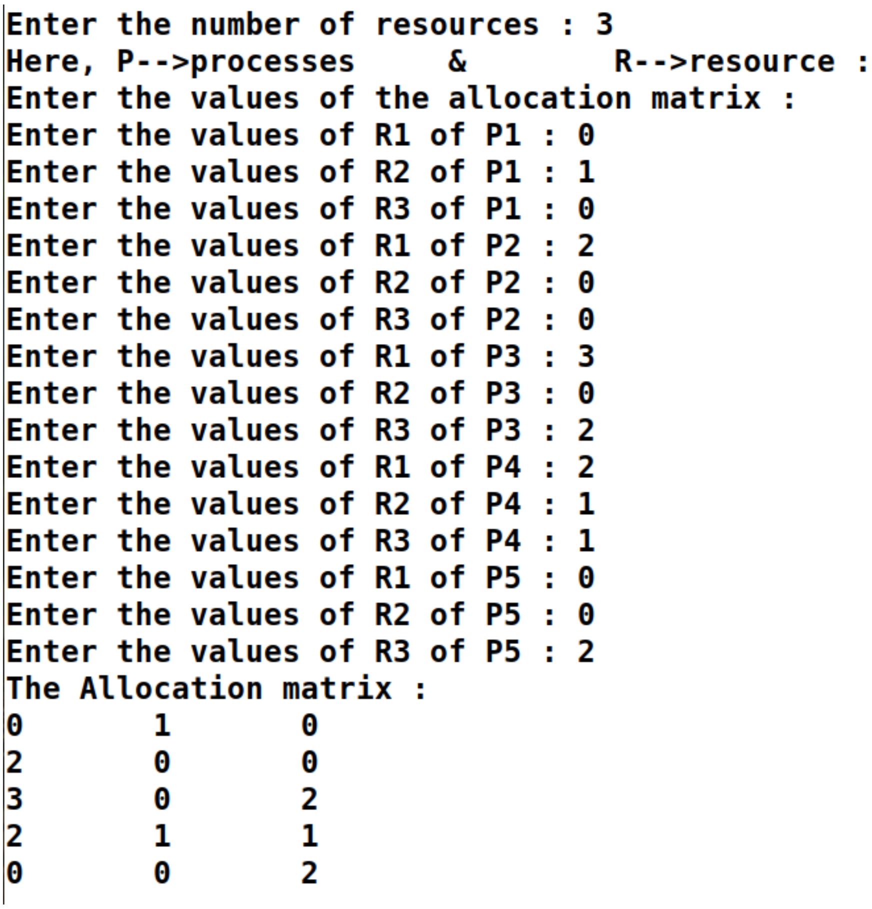


```

    int y = 0;
    for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0)
            {
                int flag = 0;
                for (j = 0; j < m; j++)
                {
                    if (need[i][j] > available_matrix[k][j])
                    {
                        flag = 1;
                         break;
                    }
                }
                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        available_matrix[k][y] += allocation_matrix[i][y];
                    f[i] = 1;
                }
            }
        }
    }
```

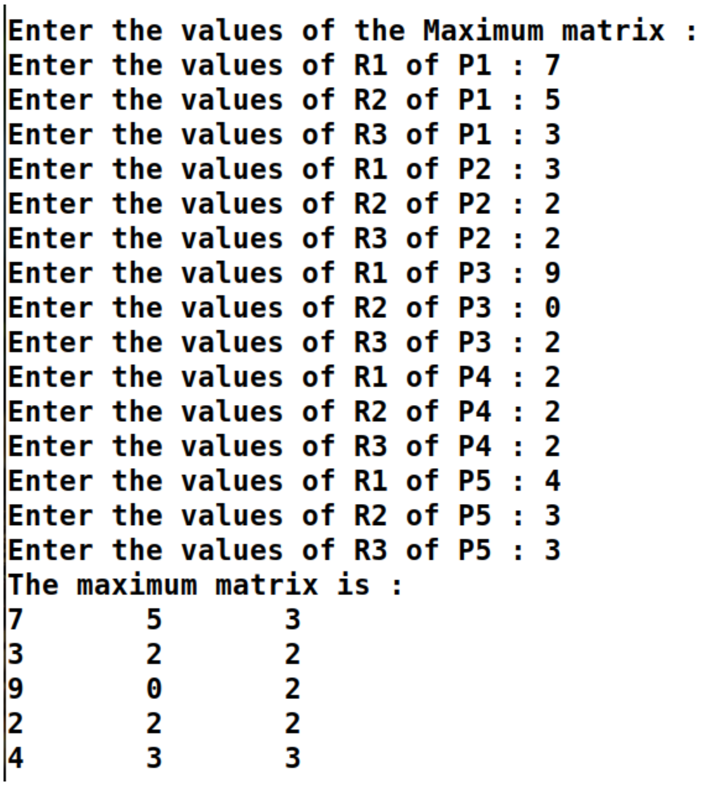


```
    }
    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);
    return (0);
}
```

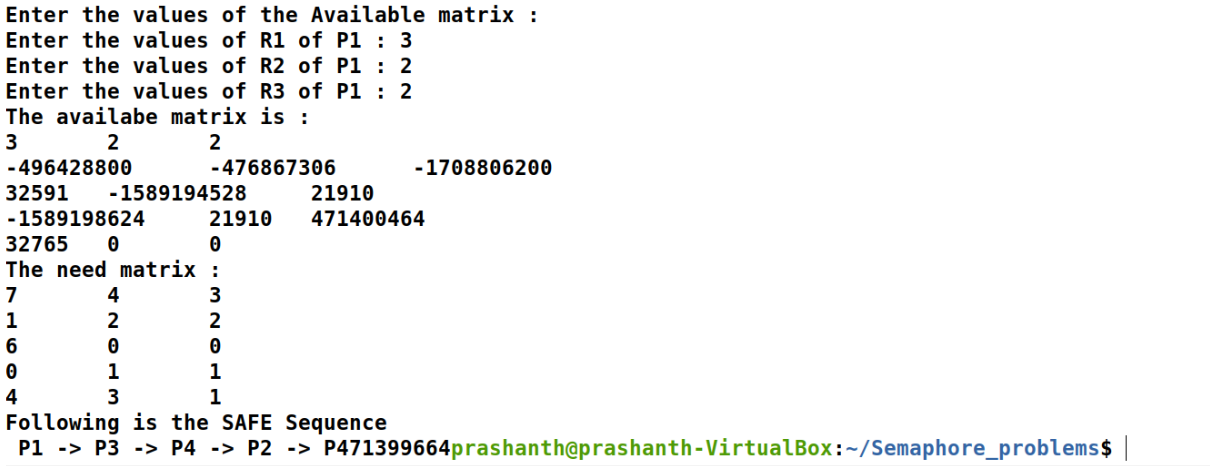
Output:



```
Enter the number of resources : 3
Here, P-->processes     &        R-->resource :
Enter the values of the allocation matrix :
Enter the values of R1 of P1 : 0
Enter the values of R2 of P1 : 1
Enter the values of R3 of P1 : 0
Enter the values of R1 of P2 : 2
Enter the values of R2 of P2 : 0
Enter the values of R3 of P2 : 0
Enter the values of R1 of P3 : 3
Enter the values of R2 of P3 : 0
Enter the values of R3 of P3 : 2
Enter the values of R1 of P4 : 2
Enter the values of R2 of P4 : 1
Enter the values of R3 of P4 : 1
Enter the values of R1 of P5 : 0
Enter the values of R2 of P5 : 0
Enter the values of R3 of P5 : 2
The Allocation matrix :
0       1       0
2       0       0
3       0       2
2       1       1
0       0       2
```



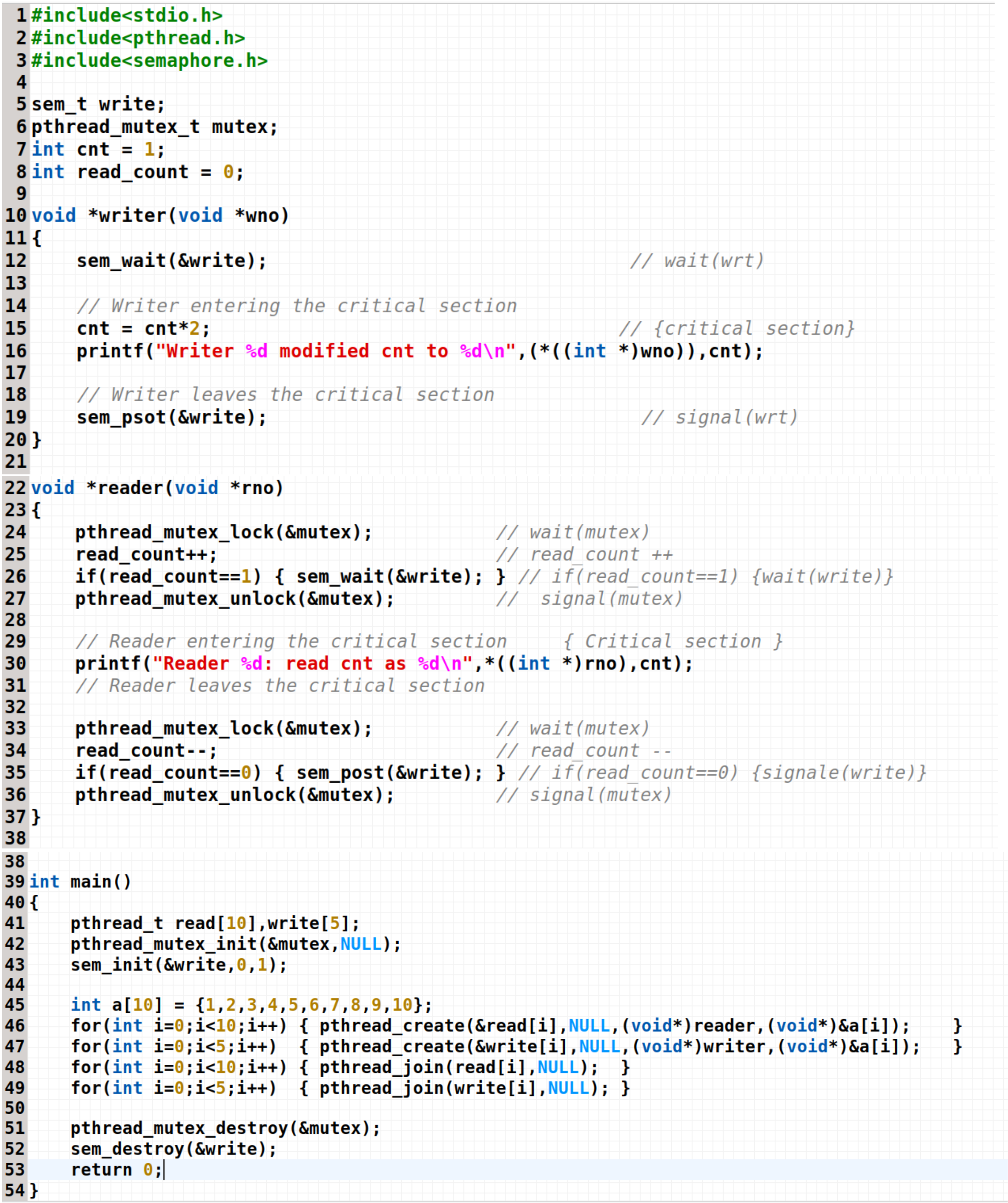
```
Enter the values of the Maximum matrix :
Enter the values of R1 of P1 : 7
Enter the values of R2 of P1 : 5
Enter the values of R3 of P1 : 3
Enter the values of R1 of P2 : 3
Enter the values of R2 of P2 : 2
Enter the values of R3 of P2 : 2
Enter the values of R1 of P3 : 9
Enter the values of R2 of P3 : 0
Enter the values of R3 of P3 : 2
Enter the values of R1 of P4 : 2
Enter the values of R2 of P4 : 2
Enter the values of R3 of P4 : 2
Enter the values of R1 of P5 : 4
Enter the values of R2 of P5 : 3
Enter the values of R3 of P5 : 3
The maximum matrix is :
7       5       3
3       2       2
9       0       2
2       2       2
4       3       3
```



```
Enter the values of the Available matrix :
Enter the values of R1 of P1 : 3
Enter the values of R2 of P1 : 2
Enter the values of R3 of P1 : 2
The availabe matrix is :
3       2       2
-496428800      -476867306      -1708806200
32591   -1589194528     21910
-1589198624     21910   471400464
32765   0       0
The need matrix :
7       4       3
1       2       2
6       0       0
0       1       1
4       3       1
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P2 -> P471399664prashanth@prashanth-VirtualBox:~/Semaphore_problems$ 
```

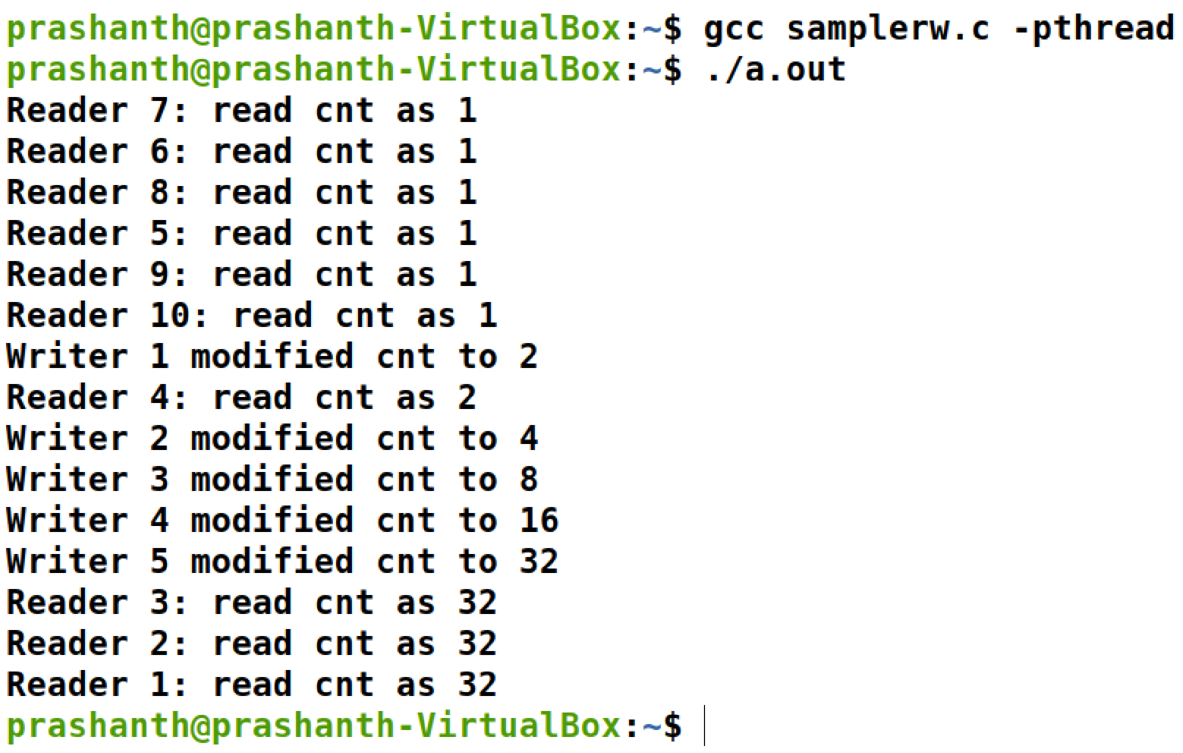
Reader and Writers problem

Code :



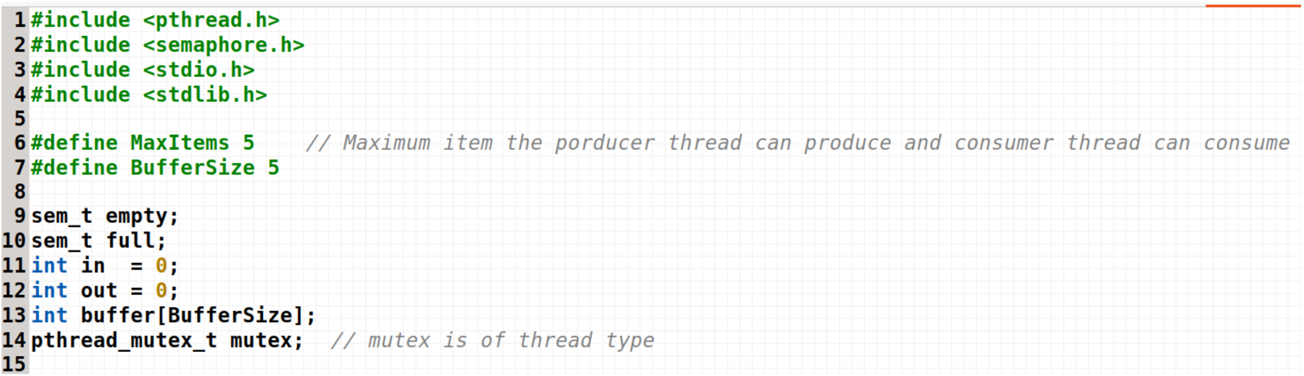
```c
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>

sem_t write;
pthread_mutex_t mutex;
int cnt = 1;
int read_count = 0;

void *writer(void *wno)
{
    sem_wait(&write);                                    // wait(wrt)

    // Writer entering the critical section
    cnt = cnt*2;                                         // {critical section}
    printf("Writer %d modified cnt to %d\n",(*((int *)wno)),cnt);

    // Writer leaves the critical section
    sem_psot(&write);                                     // signal(wrt)
}

void *reader(void *rno)
{
    pthread_mutex_lock(&mutex);              // wait(mutex)
    read_count++;                            // read_count ++
    if(read_count==1) { sem_wait(&write); } // if(read_count==1) {wait(write)}
    pthread_mutex_unlock(&mutex);            //  signal(mutex)

    // Reader entering the critical section      { Critical section }
    printf("Reader %d: read cnt as %d\n",*((int *)rno),cnt);
    // Reader leaves the critical section

    pthread_mutex_lock(&mutex);              // wait(mutex)
    read_count--;                            // read_count --
    if(read_count==0) { sem_post(&write); } // if(read_count==0) {signale(write)}
    pthread_mutex_unlock(&mutex);            // signal(mutex)
}


int main()
{
    pthread_t read[10],write[5];
    pthread_mutex_init(&mutex,NULL);
    sem_init(&write,0,1);

    int a[10] = {1,2,3,4,5,6,7,8,9,10};
    for(int i=0;i<10;i++) { pthread_create(&read[i],NULL,(void*)reader,(void*)&a[i]);    }
    for(int i=0;i<5;i++)  { pthread_create(&write[i],NULL,(void*)writer,(void*)&a[i]);   }
    for(int i=0;i<10;i++) { pthread_join(read[i],NULL);  }
    for(int i=0;i<5;i++)  { pthread_join(write[i],NULL); }

    pthread_mutex_destroy(&mutex);
    sem_destroy(&write);
    return 0;
}
```

Output:

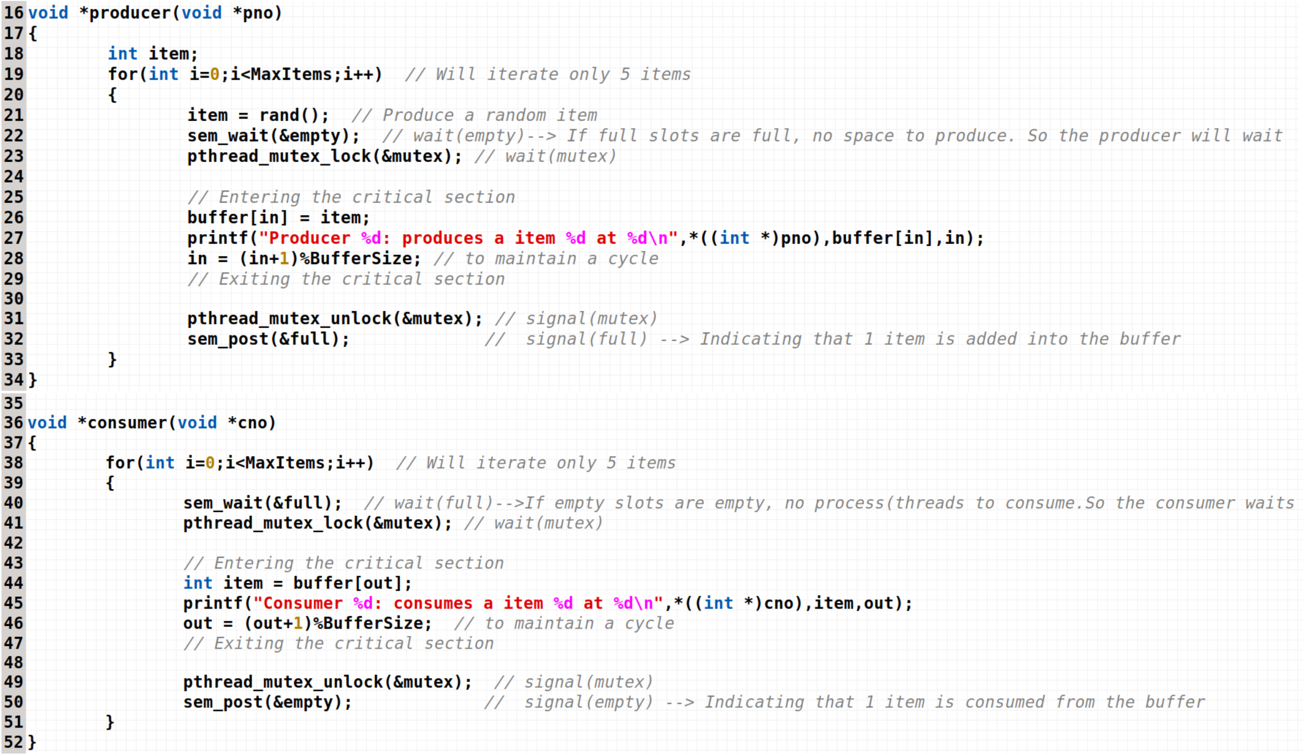


```
prashanth@prashanth-VirtualBox:~$ gcc samplerw.c -pthread
prashanth@prashanth-VirtualBox:~$ ./a.out
Reader 7: read cnt as 1
Reader 6: read cnt as 1
Reader 8: read cnt as 1
Reader 5: read cnt as 1
Reader 9: read cnt as 1
Reader 10: read cnt as 1
Writer 1 modified cnt to 2
Reader 4: read cnt as 2
Writer 2 modified cnt to 4
Writer 3 modified cnt to 8
Writer 4 modified cnt to 16
Writer 5 modified cnt to 32
Reader 3: read cnt as 32
Reader 2: read cnt as 32
Reader 1: read cnt as 32
prashanth@prashanth-VirtualBox:~$
```

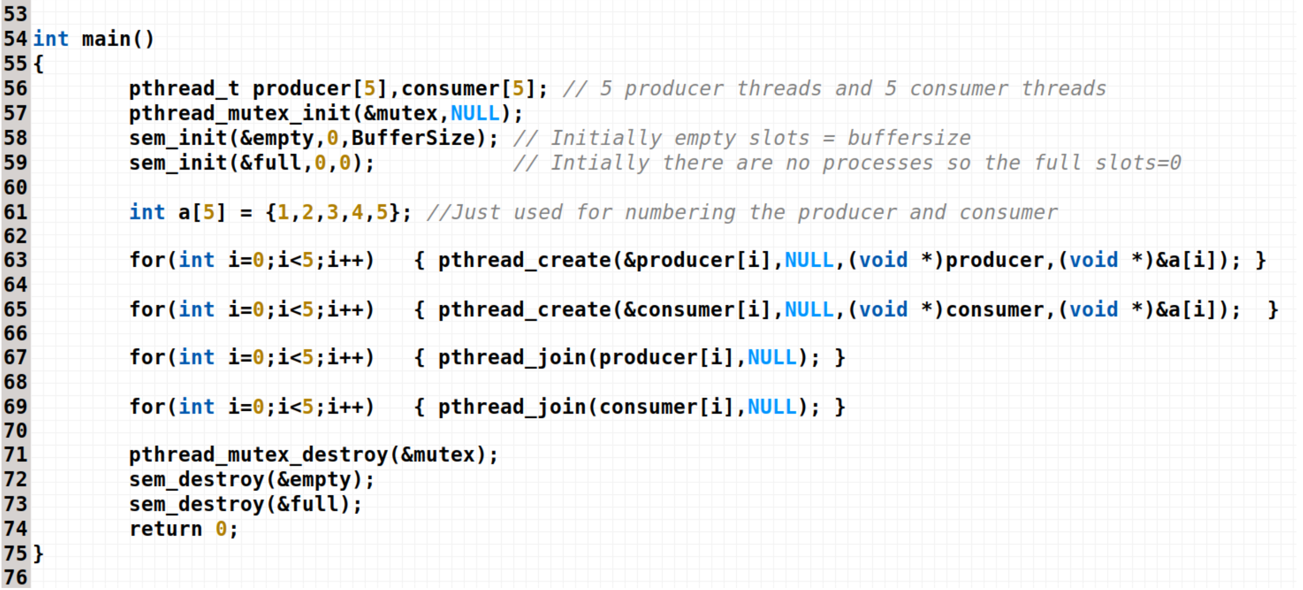
Producer-Consumer problem



```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>

#define MaxItems 5    // Maximum item the porducer thread can produce and consumer thread can consume
#define BufferSize 5

sem_t empty;
sem_t full;
int in  = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;  // mutex is of thread type

```

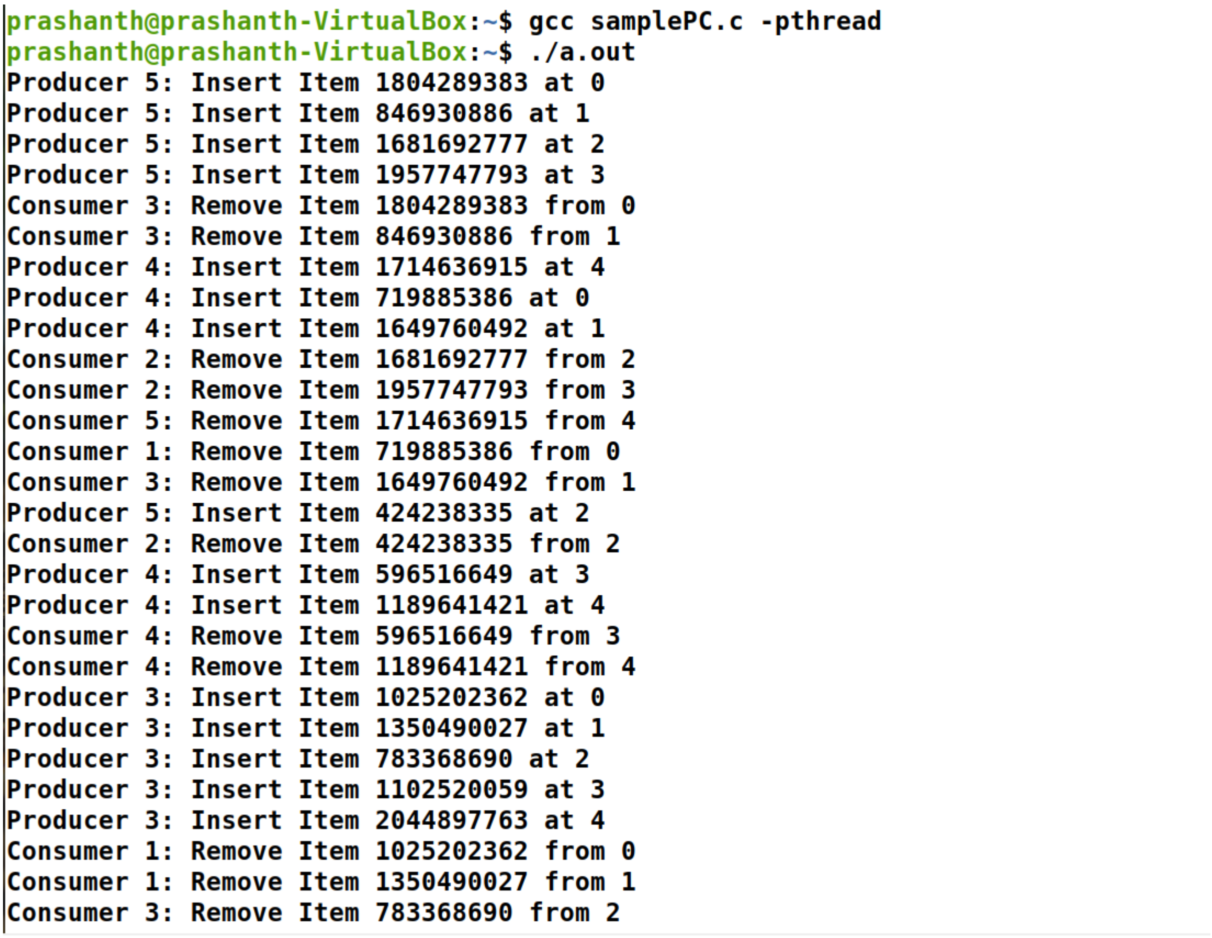


```c
void *producer(void *pno)
{
        int item;
        for(int i=0;i<MaxItems;i++)  // Will iterate only 5 items
        {
                item = rand();  // Produce a random item
                sem_wait(&empty);  // wait(empty)--> If full slots are full, no space to produce. So the producer will wait
                pthread_mutex_lock(&mutex); // wait(mutex)

                // Entering the critical section
                buffer[in] = item;
                printf("Producer %d: produces a item %d at %d\n",*((int *)pno),buffer[in],in);
                in = (in+1)%BufferSize; // to maintain a cycle
                // Exiting the critical section

                pthread_mutex_unlock(&mutex); // signal(mutex)
                sem_post(&full);             //  signal(full) --> Indicating that 1 item is added into the buffer
        }
}

void *consumer(void *cno)
{
        for(int i=0;i<MaxItems;i++)  // Will iterate only 5 items
        {
                sem_wait(&full);  // wait(full)-->If empty slots are empty, no process(threads to consume.So the consumer waits
                pthread_mutex_lock(&mutex); // wait(mutex)

                // Entering the critical section
                int item = buffer[out];
                printf("Consumer %d: consumes a item %d at %d\n",*((int *)cno),item,out);
                out = (out+1)%BufferSize;  // to maintain a cycle
                // Exiting the critical section

                pthread_mutex_unlock(&mutex);  // signal(mutex)
                sem_post(&empty);             //  signal(empty) --> Indicating that 1 item is consumed from the buffer
        }
}
```

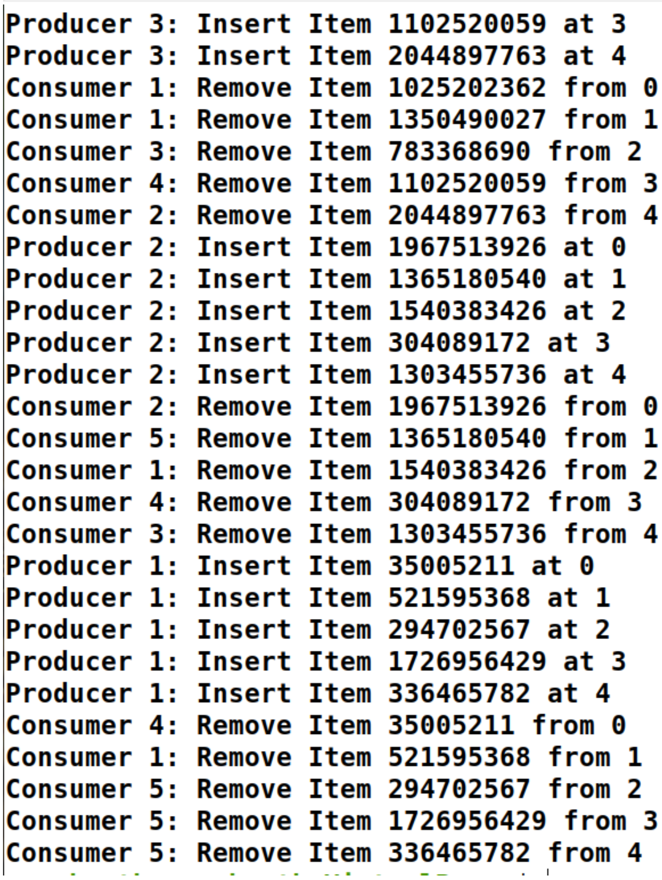


```c

int main()
{
        pthread_t producer[5],consumer[5]; // 5 producer threads and 5 consumer threads
        pthread_mutex_init(&mutex,NULL);
        sem_init(&empty,0,BufferSize); // Initially empty slots = buffersize
        sem_init(&full,0,0);           // Intially there are no processes so the full slots=0

        int a[5] = {1,2,3,4,5}; //Just used for numbering the producer and consumer

        for(int i=0;i<5;i++)   { pthread_create(&producer[i],NULL,(void *)producer,(void *)&a[i]); }

        for(int i=0;i<5;i++)   { pthread_create(&consumer[i],NULL,(void *)consumer,(void *)&a[i]);  }

        for(int i=0;i<5;i++)   { pthread_join(producer[i],NULL); }

        for(int i=0;i<5;i++)   { pthread_join(consumer[i],NULL); }

        pthread_mutex_destroy(&mutex);
        sem_destroy(&empty);
        sem_destroy(&full);
        return 0;
}

```

Output:



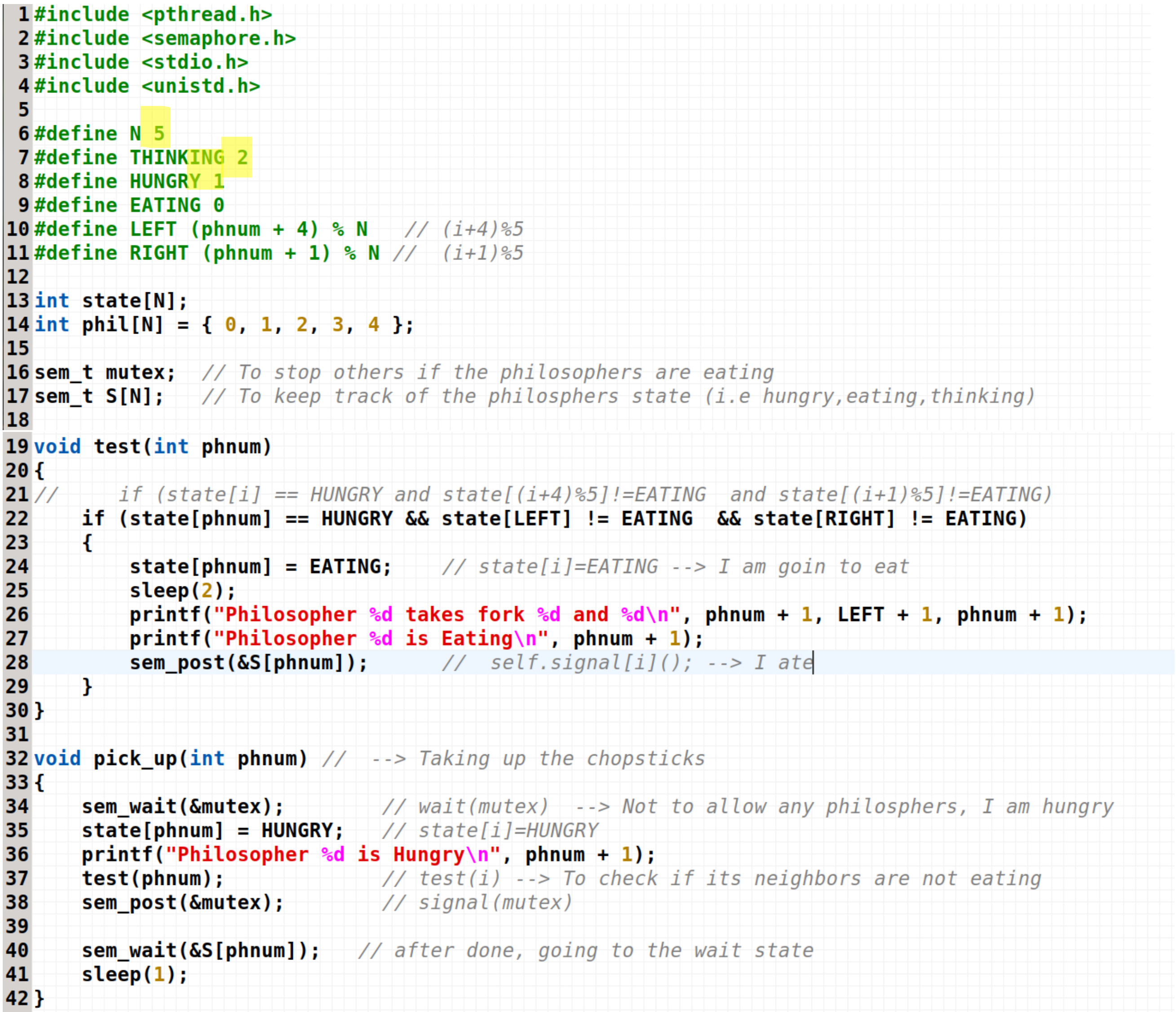
```
prashanth@prashanth-VirtualBox:~$ gcc samplePC.c -pthread
prashanth@prashanth-VirtualBox:~$ ./a.out
Producer 5: Insert Item 1804289383 at 0
Producer 5: Insert Item 846930886 at 1
Producer 5: Insert Item 1681692777 at 2
Producer 5: Insert Item 1957747793 at 3
Consumer 3: Remove Item 1804289383 from 0
Consumer 3: Remove Item 846930886 from 1
Producer 4: Insert Item 1714636915 at 4
Producer 4: Insert Item 719885386 at 0
Producer 4: Insert Item 1649760492 at 1
Consumer 2: Remove Item 1681692777 from 2
Consumer 2: Remove Item 1957747793 from 3
Consumer 5: Remove Item 1714636915 from 4
Consumer 1: Remove Item 719885386 from 0
Consumer 3: Remove Item 1649760492 from 1
Producer 5: Insert Item 424238335 at 2
Consumer 2: Remove Item 424238335 from 2
Producer 4: Insert Item 596516649 at 3
Producer 4: Insert Item 1189641421 at 4
Consumer 4: Remove Item 596516649 from 3
Consumer 4: Remove Item 1189641421 from 4
Producer 3: Insert Item 1025202362 at 0
Producer 3: Insert Item 1350490027 at 1
Producer 3: Insert Item 783368690 at 2
Producer 3: Insert Item 1102520059 at 3
Producer 3: Insert Item 2044897763 at 4
Consumer 1: Remove Item 1025202362 from 0
Consumer 1: Remove Item 1350490027 from 1
Consumer 3: Remove Item 783368690 from 2
```



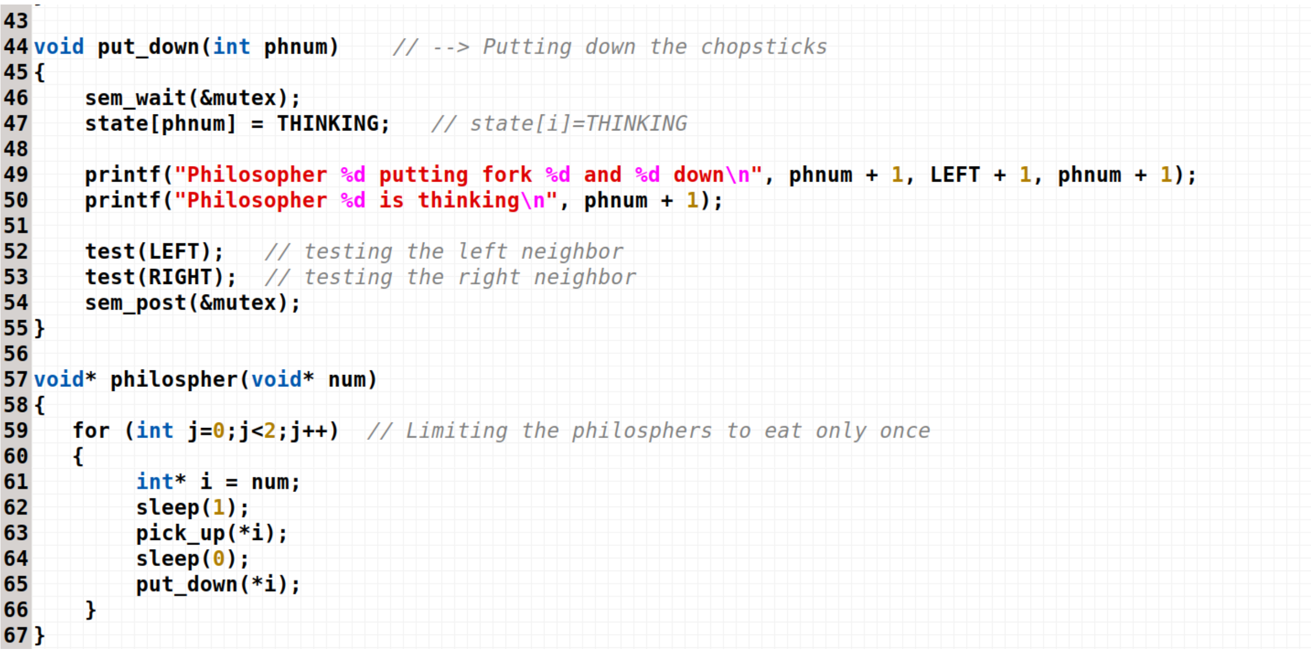
```
Producer 3: Insert Item 1102520059 at 3
Producer 3: Insert Item 2044897763 at 4
Consumer 1: Remove Item 1025202362 from 0
Consumer 1: Remove Item 1350490027 from 1
Consumer 3: Remove Item 783368690 from 2
Consumer 4: Remove Item 1102520059 from 3
Consumer 2: Remove Item 2044897763 from 4
Producer 2: Insert Item 1967513926 at 0
Producer 2: Insert Item 1365180540 at 1
Producer 2: Insert Item 1540383426 at 2
Producer 2: Insert Item 304089172 at 3
Producer 2: Insert Item 1303455736 at 4
Consumer 2: Remove Item 1967513926 from 0
Consumer 5: Remove Item 1365180540 from 1
Consumer 1: Remove Item 1540383426 from 2
Consumer 4: Remove Item 304089172 from 3
Consumer 3: Remove Item 1303455736 from 4
Producer 1: Insert Item 35005211 at 0
Producer 1: Insert Item 521595368 at 1
Producer 1: Insert Item 294702567 at 2
Producer 1: Insert Item 1726956429 at 3
Producer 1: Insert Item 336465782 at 4
Consumer 4: Remove Item 35005211 from 0
Consumer 1: Remove Item 521595368 from 1
Consumer 5: Remove Item 294702567 from 2
Consumer 5: Remove Item 1726956429 from 3
Consumer 5: Remove Item 336465782 from 4
```

Dining Phosphors Problem

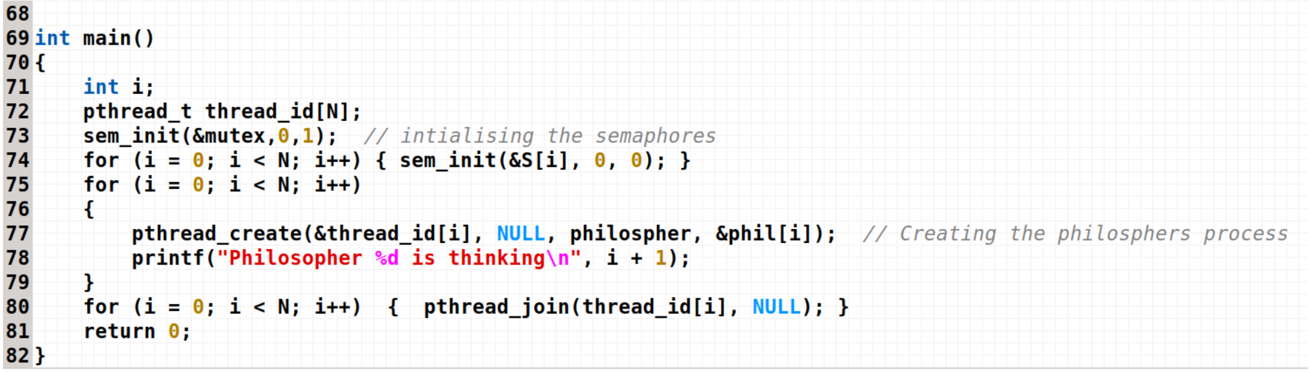
Code:



```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N   // (i+4)%5
#define RIGHT (phnum + 1) % N //  (i+1)%5

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;  // To stop others if the philosophers are eating
sem_t S[N];   // To keep track of the philosphers state (i.e hungry,eating,thinking)

void test(int phnum)
{
//     if (state[i] == HUNGRY and state[(i+4)%5]!=EATING  and state[(i+1)%5]!=EATING)
    if (state[phnum] == HUNGRY && state[LEFT] != EATING  && state[RIGHT] != EATING)
    {
        state[phnum] = EATING;    // state[i]=EATING --> I am goin to eat
        sleep(2);
        printf("Philosopher %d takes fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is Eating\n", phnum + 1);
        sem_post(&S[phnum]);      //  self.signal[i](); --> I ate
    }
}

void pick_up(int phnum) //  --> Taking up the chopsticks
{
    sem_wait(&mutex);        // wait(mutex)  --> Not to allow any philosphers, I am hungry
    state[phnum] = HUNGRY;   // state[i]=HUNGRY
    printf("Philosopher %d is Hungry\n", phnum + 1);
    test(phnum);             // test(i) --> To check if its neighbors are not eating
    sem_post(&mutex);        // signal(mutex)

    sem_wait(&S[phnum]);   // after done, going to the wait state
    sleep(1);
}
```

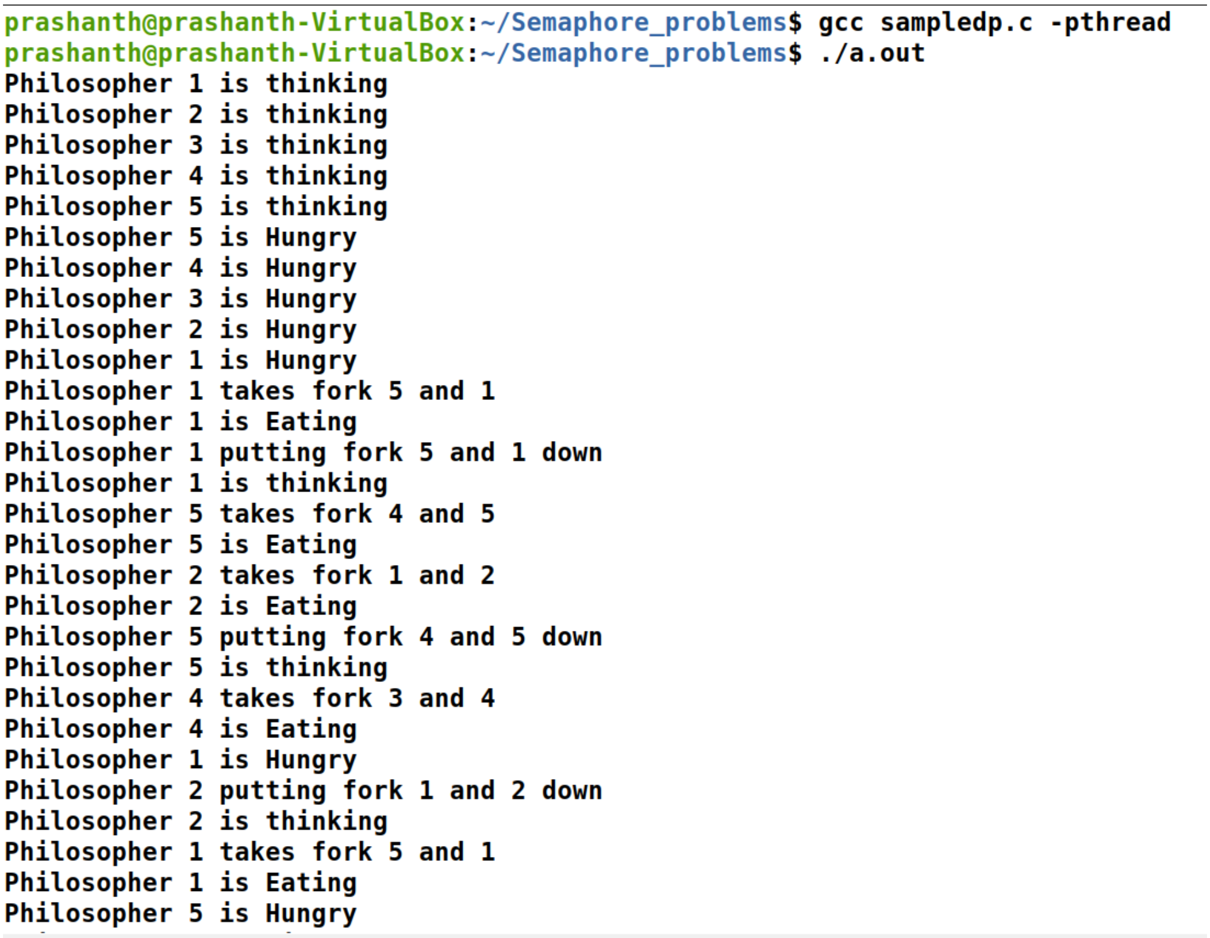


```c

void put_down(int phnum)    // --> Putting down the chopsticks
{
    sem_wait(&mutex);
    state[phnum] = THINKING;   // state[i]=THINKING

    printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);

    test(LEFT);   // testing the left neighbor
    test(RIGHT);  // testing the right neighbor
    sem_post(&mutex);
}

void* philospher(void* num)
{
   for (int j=0;j<2;j++)  // Limiting the philosphers to eat only once
   {
        int* i = num;
        sleep(1);
        pick_up(*i);
        sleep(0);
        put_down(*i);
    }
}
```

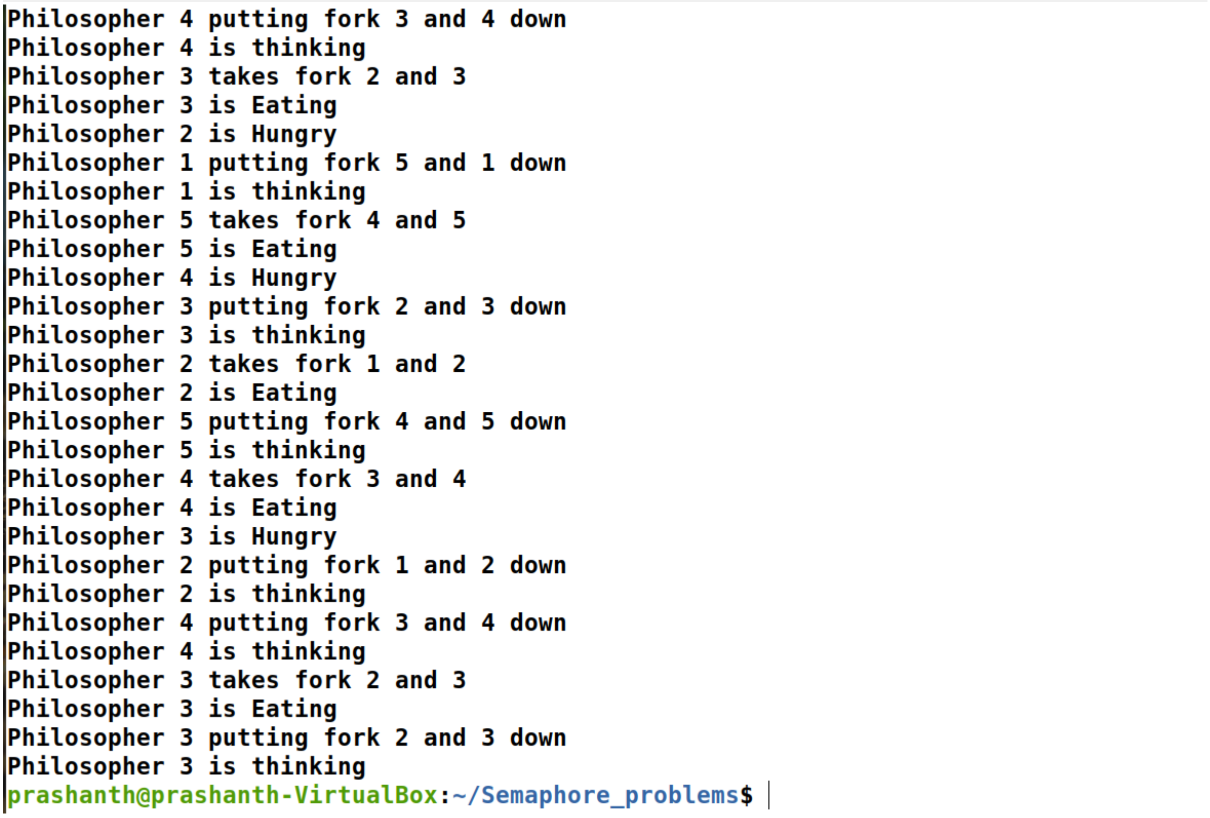


```c

int main()
{
    int i;
    pthread_t thread_id[N];
    sem_init(&mutex,0,1);  // intialising the semaphores
    for (i = 0; i < N; i++) { sem_init(&S[i], 0, 0); }
    for (i = 0; i < N; i++)
    {
        pthread_create(&thread_id[i], NULL, philospher, &phil[i]);  // Creating the philosphers process
        printf("Philosopher %d is thinking\n", i + 1);
    }
    for (i = 0; i < N; i++)  {  pthread_join(thread_id[i], NULL); }
    return 0;
}
```

Output:

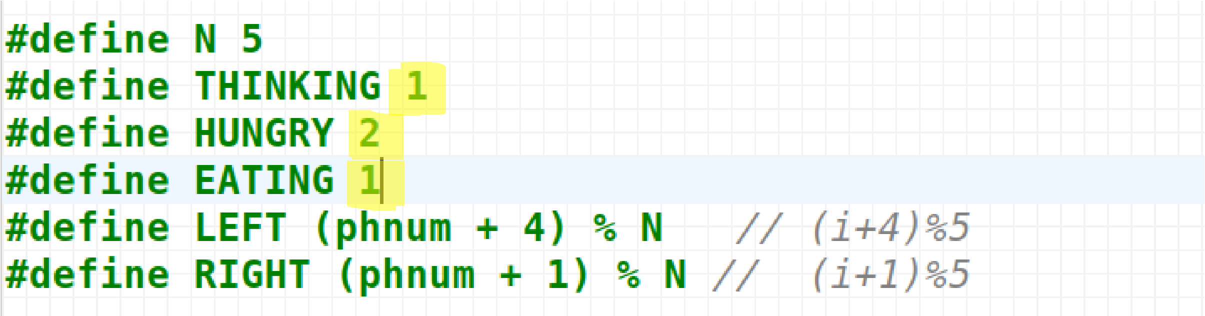


```
prashanth@prashanth-VirtualBox:~/Semaphore_problems$ gcc sampledp.c -pthread
prashanth@prashanth-VirtualBox:~/Semaphore_problems$ ./a.out
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 3 is Hungry
Philosopher 2 is Hungry
Philosopher 1 is Hungry
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
```

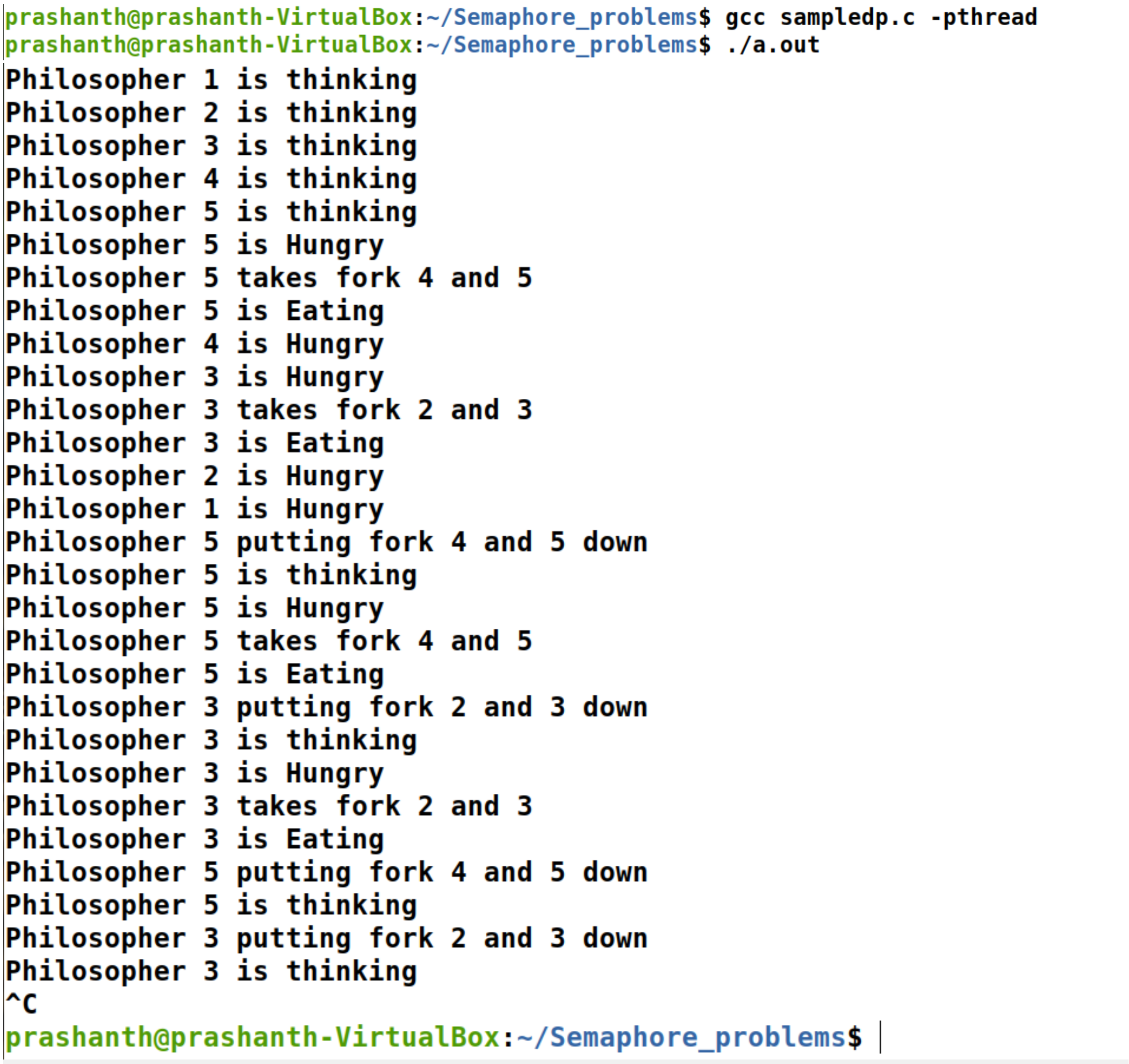


```
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
prashanth@prashanth-VirtualBox:~/Semaphore_problems$
```

After changing the initial conditions → Leading to starvation



```
#define N 5
#define THINKING 1
#define HUNGRY 2
#define EATING 1
#define LEFT (phnum + 4) % N    // (i+4)%5
#define RIGHT (phnum + 1) % N //   (i+1)%5
```



```
prashanth@prashanth-VirtualBox:~/Semaphore_problems$ gcc sampledp.c -pthread
prashanth@prashanth-VirtualBox:~/Semaphore_problems$ ./a.out
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 is Hungry
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 3 is Hungry
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
^C
prashanth@prashanth-VirtualBox:~/Semaphore_problems$ 
```