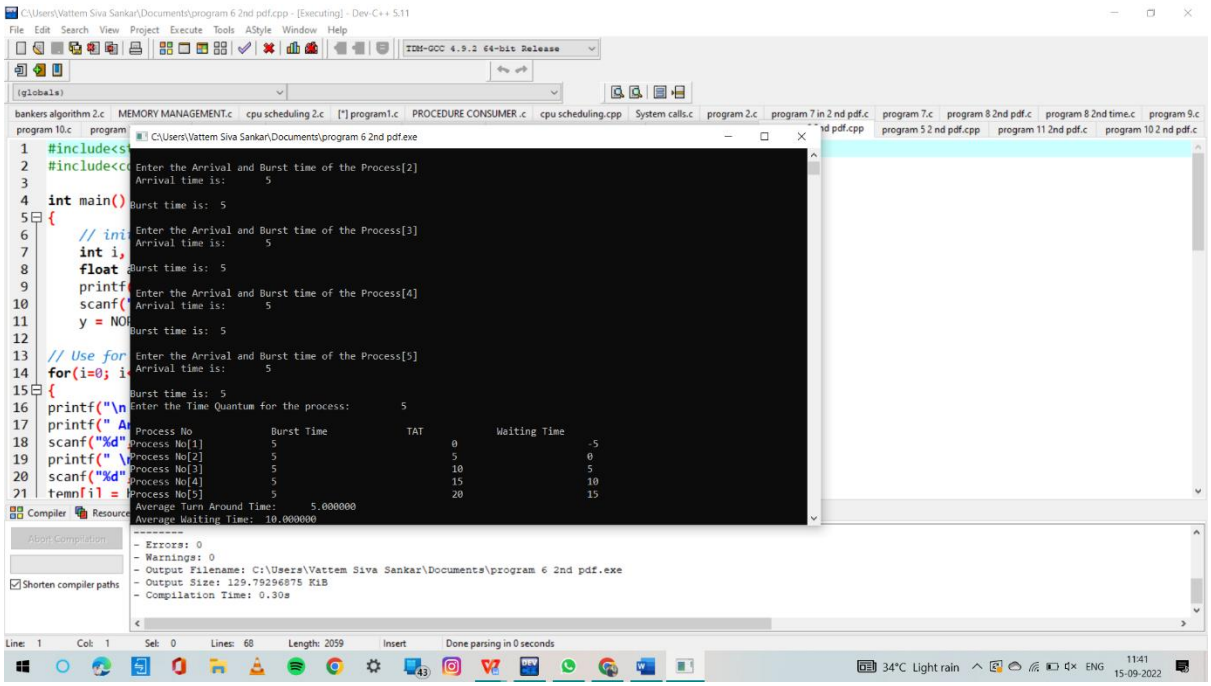
1.Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.

```
#include<stdio.h>

int main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Enter number of process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("p%d:",i+1);
      scanf("%d",&bt[i]);
      p[i]=i+1;          //contains process number
   }

   //sorting burst time in ascending order using selection sort
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(bt[j]<bt[pos])
            pos=j;
      }

      temp=bt[i];
      bt[i]=bt[pos];
```

```
    bt[pos]=temp;

    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}

wt[0]=0;          //waiting time for first process will be zero

//calculate waiting time
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];

    total+=wt[i];
}

avg_wt=(float)total/n;      //average waiting time
total=0;

printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];     //calculate turnaround time
    total+=tat[i];
    printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}

avg_tat=(float)total/n;     //average turnaround time
```

```
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

OUTPUT:



6.Construct a C program to simulate Round Robin scheduling algorithm with C.

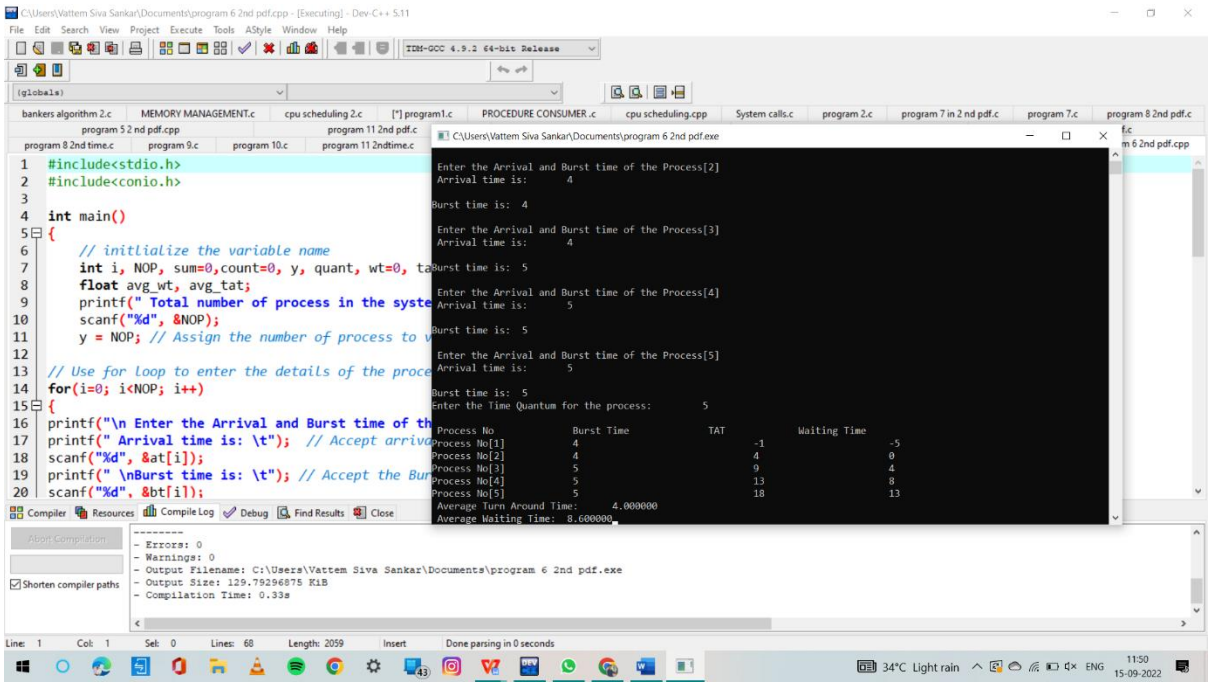
```
#include<stdio.h>

#include<conio.h>



int main()

{

   // initlialize the variable name

   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];

   float avg_wt, avg_tat;

   printf(" Total number of process in the system: ");
```

```
    scanf("%d", &NOP);
    y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t");  // Accept arrival time
scanf("%d", &at[i]);
printf(" \nBurst time is: \t"); // Accept the Burst time
scanf("%d", &bt[i]);
temp[i] = bt[i]; // store the burst time in temp array
}
// Accept the Time qunat
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
// Display the process No, burst time, Turn Around Time and the waiting time
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
```

```
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
   sum = sum + temp[i];
   temp[i] = 0;
   count=1;
   }
   else if(temp[i] > 0)
   {
      temp[i] = temp[i] - quant;
      sum = sum + quant;
   }
   if(temp[i]==0 && count==1)
   {
      y--; //decrement the process no.
      printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-
bt[i]);
      wt = wt+sum-at[i]-bt[i];
      tat = tat+sum-at[i];
      count =0;
   }
   if(i==NOP-1)
```

```
    {
        i=0;
    }
    else if(at[i+1]<=sum)
    {
        i++;
    }
    else
    {
        i=0;
    }
}
// represents the average waiting time and Turn Around time
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();
}
```

OUTPUT:



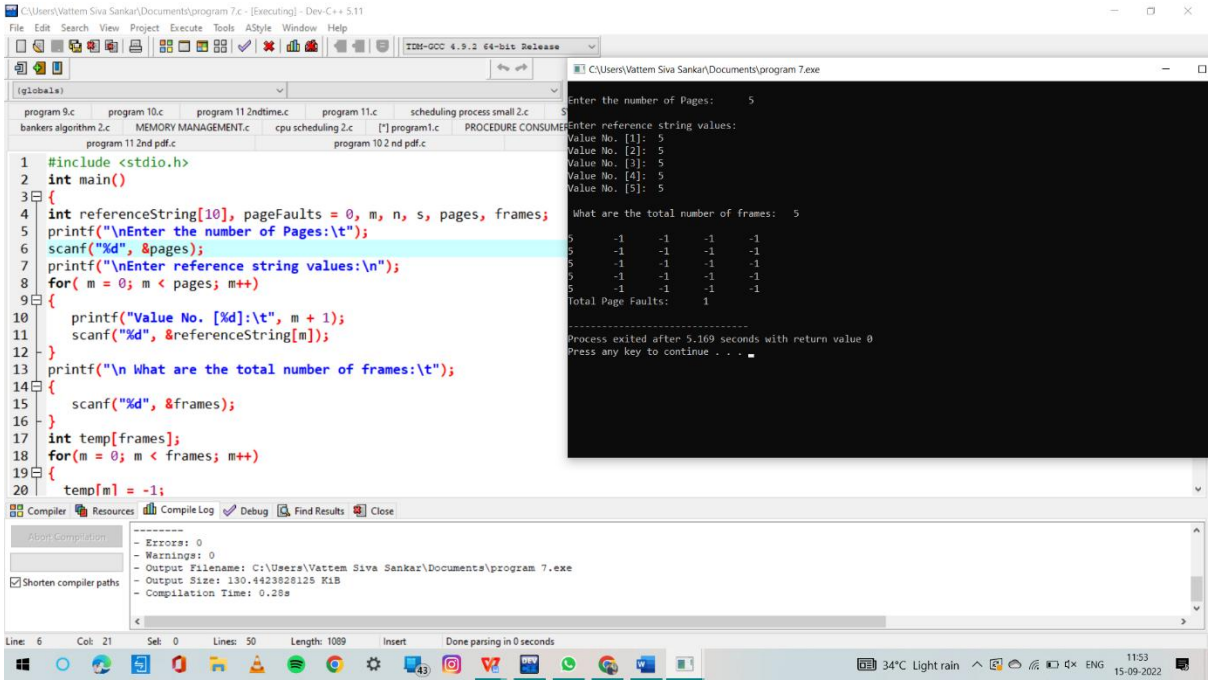
7. Illustrate the concept of inter-process communication using shared memory with a C program.

```
#include <stdio.h>

int main()

{

int referenceString[10], pageFaults = 0, m, n, s, pages, frames;

printf("\nEnter the number of Pages:\t");

scanf("%d", &pages);

printf("\nEnter reference string values:\n");

for( m = 0; m < pages; m++)

{

   printf("Value No. [%d]:\t", m + 1);

   scanf("%d", &referenceString[m]);
```

```
}
printf("\n What are the total number of frames:\t");
{
   scanf("%d", &frames);
}
int temp[frames];
for(m = 0; m < frames; m++)
{
  temp[m] = -1;
}
for(m = 0; m < pages; m++)
{
  s = 0;
  for(n = 0; n < frames; n++)
   {
      if(referenceString[m] == temp[n])
         {
            s++;
            pageFaults--;
         }
```

```
    }
    pageFaults++;
    if((pageFaults <= frames) && (s == 0))
      {
       temp[m] = referenceString[m];
      }
    else if(s == 0)
      {
       temp[(pageFaults - 1) % frames] = referenceString[m];
      }
      printf("\n");
      for(n = 0; n < frames; n++)
       {
         printf("%d\t", temp[n]);
       }
}
printf("\nTotal Page Faults:\t%d\n", pageFaults);
return 0;
}
```

OUTPUT:

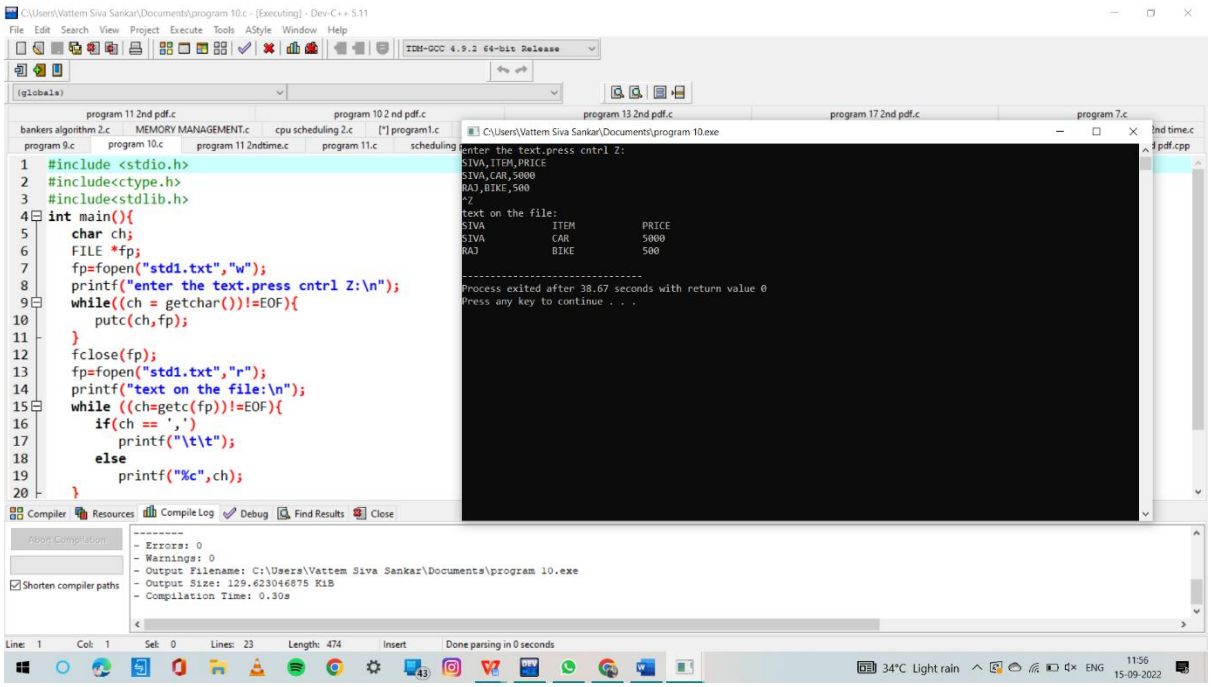


10. Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a C program to simulate the file allocation strategy.

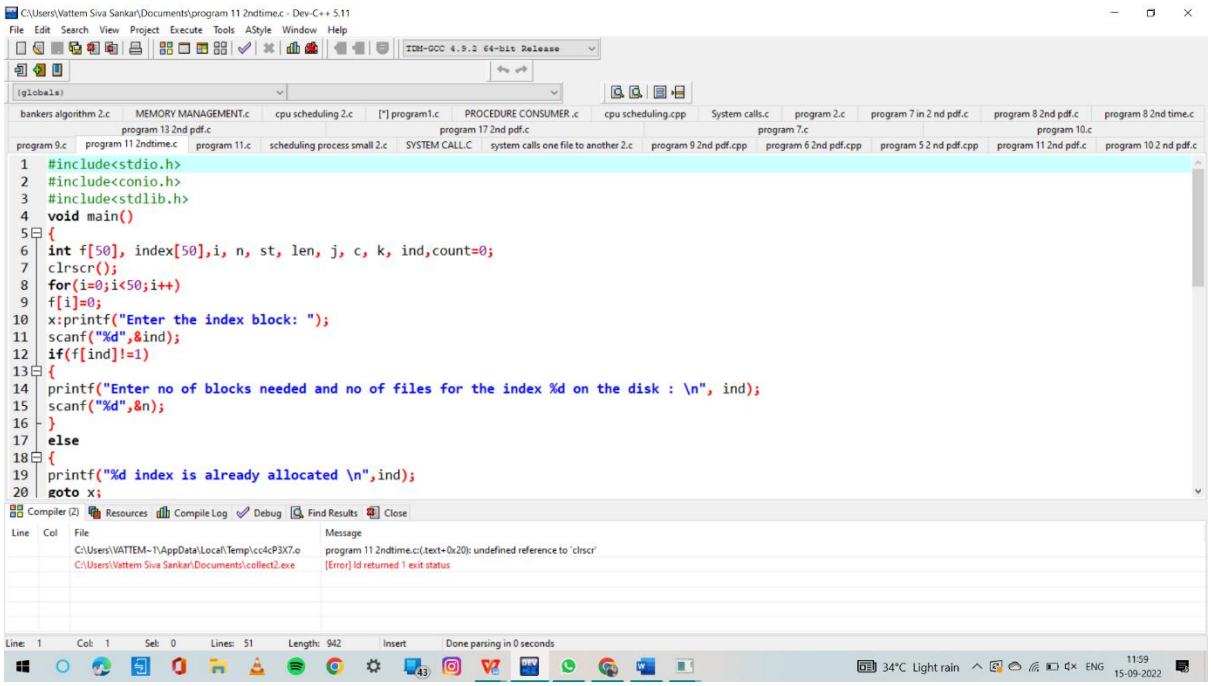
```
#include <stdio.h>
#include<ctype.h>
#include<stdlib.h>
int main(){
  char ch;
  FILE *fp;
  fp=fopen("std1.txt","w");
  printf("enter the text.press cntrl Z:\n");
  while((ch = getchar())!=EOF){
    putc(ch,fp);
  }
  fclose(fp);
  fp=fopen("std1.txt","r");
  printf("text on the file:\n");
  while ((ch=getc(fp))!=EOF){
    if(ch == ',')
```

```
        printf("\t\t");
      else
        printf("%c",ch);
   }
   fclose(fp);
   return 0;
}
```

OUTPUT:



11. Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a C program to simulate the file allocation strategy.



9. Design a C program to simulate the concept of Dining-Philosophers problem

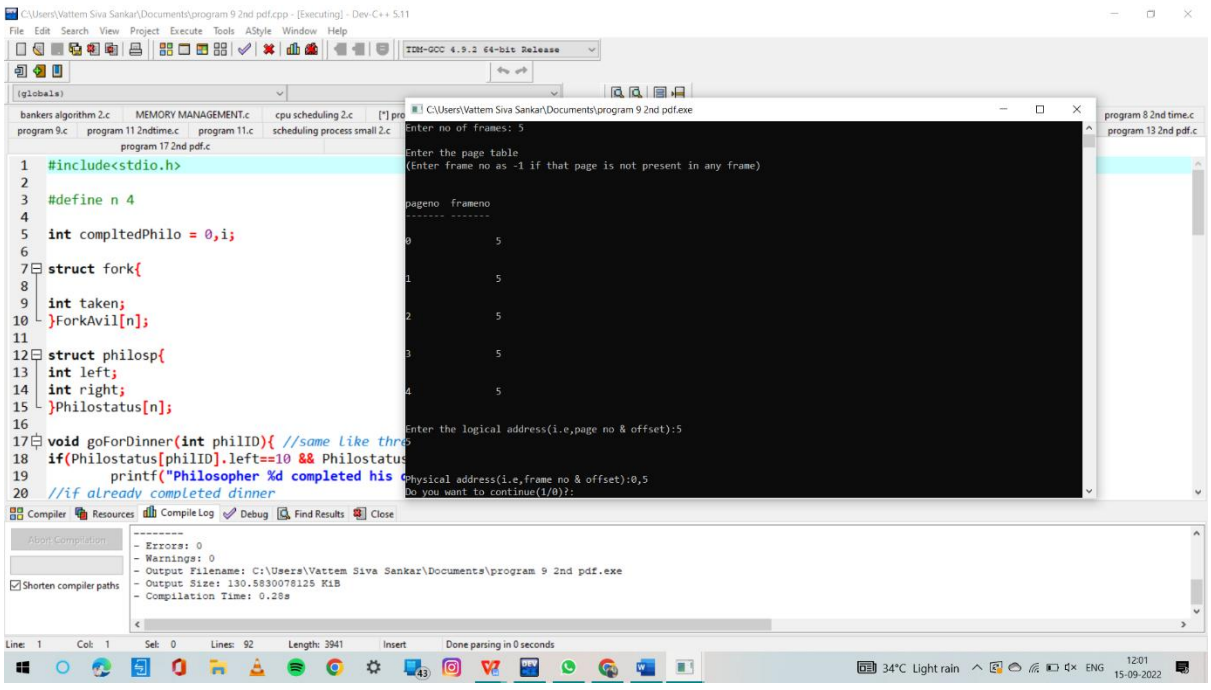
```
#include<stdio.h>

#define n 4

int compltedPhilo = 0,i;

struct fork{

int taken;
}ForkAvil[n];

struct philosp{
int left;
int right;
}Philostatus[n];

void goForDinner(int philID){ //same like threads concept here cases implemented
if(Philostatus[philID].left==10 && Philostatus[philID].right==10)
```

```
        printf("Philosopher %d completed his dinner\n",philID+1);
//if already completed dinner
else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){
            //if just taken two forks
            printf("Philosopher %d completed his dinner\n",philID+1);

            Philostatus[philID].left = Philostatus[philID].right = 10; //remembering that he completed dinner by assigning value 10
            int otherFork = philID-1;

            if(otherFork== -1)
               otherFork=(n-1);

            ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0; //releasing forks
            printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);
            compltedPhilo++;
        }
        else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){ //left already taken, trying for right fork
            if(philID==(n-1)){
               if(ForkAvil[philID].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION
                  ForkAvil[philID].taken = Philostatus[philID].right = 1;
                  printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);
               }else{
                  printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);
               }
            }else{ //except last philosopher case
               int dupphilID = philID;
               philID-=1;

               if(philID== -1)
```

```
                philID=(n-1);


            if(ForkAvil[philID].taken == 0){
                ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;
                printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);
            }else{
                printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);
            }
          }
        }
        else if(Philostatus[philID].left==0){ //nothing taken yet
            if(philID==(n-1)){
                if(ForkAvil[philID-1].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST
PHILOSOPHER TRYING IN reverse DIRECTION
                    ForkAvil[philID-1].taken = Philostatus[philID].left = 1;
                    printf("Fork %d taken by philosopher %d\n",philID,philID+1);
                }else{
                    printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
                }
            }else{ //except last philosopher case
                if(ForkAvil[philID].taken == 0){
                    ForkAvil[philID].taken = Philostatus[philID].left = 1;
                    printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
                }else{
                    printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
                }
            }
      }else{}
}


int main(){
```

```
for(i=0;i<n;i++)
        ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;


while(compltedPhilo<n){
/* Observe here carefully, while loop will run until all philosophers complete dinner
Actually problem of deadlock occur only thy try to take at same time
This for loop will say that they are trying at same time. And remaining status will print by go for dinner function
*/
for(i=0;i<n;i++)
            goForDinner(i);
printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);
}


return 0;
}
```

OUTPUT:



10. Construct a C program for implementation of memory allocation using first fit strategy.

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

```
void main()
{
int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

for(i = 0; i < 10; i++)
{
flags[i] = 0;
allocation[i] = -1;
}
printf("Enter no. of blocks: ");
scanf("%d", &bno);
printf("\nEnter size of each block: ");
for(i = 0; i < bno; i++)
scanf("%d", &bsize[i]);

printf("\nEnter no. of processes: ");
scanf("%d", &pno);
printf("\nEnter size of each process: ");
for(i = 0; i < pno; i++)
scanf("%d", &psize[i]);
for(i = 0; i < pno; i++)        //allocation as per first fit
for(j = 0; j < bno; j++)
if(flags[j] == 0 && bsize[j] >= psize[i])
{
allocation[j] = i;
flags[j] = 1;
break;
}
//display allocation details
printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
```

```
for(i = 0; i < bno; i++)
{
printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
if(flags[i] == 1)
printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
else
printf("Not allocated");
}
}
```

OUTPUT: