Single level dic

```
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
int main()
{
int i,ch;
char f[30];
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n1. Create File\t2. Delete File\t3. Search File
\n 4. Display Files\t5. Exit\nEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2: printf("\nEnter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]); break; } }
if(i==dir.fcnt) printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\nEnter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\nDirectory Empty");
else
{
printf("\nThe Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default:exit(0);
}
}
}
```

Two level dic

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct
{
   char dname[10],fname[10][10];
   int fcnt;
}dir[10];
int main()
{
   int i,ch,dcnt,k;
   char f[30], d[30];
   // clrscr();
   dcnt=0;
   while(1)
   {
      printf("\n\n1. Create Directory\t2. Create File\t3.
Delete File");
      printf("\n4. Search File\t\t5. Display\t6.
Exit\nEnter your choice --");
   scanf("%d",&ch);
   switch(ch)
   {
      case 1: printf("\nEnter name of directory -- ");
            scanf("%s", dir[dcnt].dname);
            dir[dcnt].fcnt=0;
            dcnt++;
            printf("Directory created");
         break;
      case 2: printf("\nEnter name of the directory -- ");
            scanf("%s",d);
            for(i=0;i<dcnt;i++)
               if(strcmp(d,dir[i].dname)==0)
               {
                  printf("Enter name of the file -- ");
                  scanf("%s",dir[i].fname[dir[i].fcnt]);
                  dir[i].fcnt++;
                  printf("File created");
                  break;
               }
            if(i==dcnt)
               printf("Directory %s not found",d);
            break;
      case 3: printf("\nEnter name of the directory -- ");
            scanf("%s",d);
            for(i=0;i<dcnt;i++)
            {
               if(strcmp(d,dir[i].dname)==0)
               {
                  printf("Enter name of the file -- ");
                  scanf("%s",f);
                  for(k=0;k<dir[i].fcnt;k++)
                  {
                     if(strcmp(f, dir[i].fname[k])==0)
                     {
                        printf("File %s is deleted ",f);
                        dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
                        goto jmp;
                     }
                  }
               printf("File %s not found",f);
               goto jmp;
            }
         }
         printf("Directory %s not found",d);
         jmp : break;
      case 4: printf("\nEnter name of the directory -- ");
         scanf("%s",d);
         for(i=0;i<dcnt;i++)
         {
            if(strcmp(d,dir[i].dname)==0)
            {
               printf("Enter the name of the file -- ");
               scanf("%s",f);
               for(k=0;k<dir[i].fcnt;k++)
               {
                  if(strcmp(f, dir[i].fname[k])==0)
                  {
                     printf("File %s is found ",f);
                     goto jmp1;
                  }
               }
               printf("File %s not found",f);
               goto jmp1;
            }
         }
         printf("Directory %s not found",d);
         jmp1: break;
      case 5: if(dcnt==0)
            printf("\nNo Directory's ");
         else
         {
            printf("\nDirectory\tFiles");
            for(i=0;i<dcnt;i++)
            {
               printf("\n%s\t\t",dir[i].dname);
               for(k=0;k<dir[i].fcnt;k++)
               printf("\t%s",dir[i].fname[k]);
            }
         }
         break;
      default:exit(0);
   }
```

THREAD

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

#define MAX 10

int count = 0;
sem_t mutex;

void *thread1(void *arg)
{
   int i;
   sem_wait(&mutex);
   printf("Thread1 started\n");
   for (i = 0; i < MAX; i++)
   {
      count++;
      printf("Thread1 count: %d\n", count * 2);
   }
   sem_post(&mutex);
}

void *thread2(void *arg)
{
   int i;
   sem_wait(&mutex);
   printf("Thread2 started\n");
   for (i = 0; i < MAX; i++)
   {
      count++;
      printf("Thread2 count: %d\n", count * 5);
   }
   sem_post(&mutex);
}

int main()
{
   pthread_t t1, t2;
   sem_init(&mutex, 0, 1);

   pthread_create(&t1, NULL, thread1, NULL);
   pthread_create(&t2, NULL, thread2, NULL);

   pthread_join(t1, NULL);
   pthread_join(t2, NULL);

   sem_destroy(&mutex);

   return 0;
}
```

SJF NON

```
#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;
   }
  //sorting of burst times
   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;
   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;
   total=0;
   printf("\nProcess\t    Burst Time    \tWaiting
Time\tTurnaround Time");
   for(i=0;i<n;i++)
   {
      tat[i]=bt[i]+wt[i];
      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;
   printf("\n\nAverage Waiting Time=%f",avg_wt);
   printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

PRIORITY NON PRE

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

#define MAX_PROCESSES 10

typedef struct {
   int processID;
   int burstTime;
   int priority;
   int turnaroundTime;
   int waitingTime;
} Process;

void priorityNonPreemptive(Process processes[], int
numOfProcesses) {
   int completedProcesses = 0;
   int currentTime = 0;
   int highestPriorityIndex;

   while (completedProcesses < numOfProcesses) {
      highestPriorityIndex = -1;
      int highestPriority = INT_MAX;

      // Find the process with the highest priority
      for (int i = 0; i < numOfProcesses; i++) {
         if (processes[i].burstTime > 0 &&
processes[i].priority < highestPriority) {
            highestPriorityIndex = i;
            highestPriority = processes[i].priority;
         }
      }

      if (highestPriorityIndex == -1) {
         currentTime++;
         continue;
      }

      // Execute the process with the highest priority
      processes[highestPriorityIndex].burstTime--;
      currentTime++;

      // Check if the process has completed
      if (processes[highestPriorityIndex].burstTime ==
0) {
         completedProcesses++;
processes[highestPriorityIndex].turnaroundTime =
currentTime;
         processes[highestPriorityIndex].waitingTime =
processes[highestPriorityIndex].turnaroundTime -
processes[highestPriorityIndex].burstTime;
printf("Process %d completed at time %d\n",
processes[highestPriorityIndex].processID,
currentTime);
      }
   }
}

int main() {
   Process processes[MAX_PROCESSES];
   int numOfProcesses;
   int i;
   float avgTurnaroundTime = 0, avgWaitingTime = 0;

   printf("Enter the number of processes: ");
   scanf("%d", &numOfProcesses);

   printf("Enter the burst time and priority for each
process:\n");
   for (i = 0; i < numOfProcesses; i++) {
      printf("Process %d\n", i + 1);
      printf("Burst time: ");
      scanf("%d", &processes[i].burstTime);
      printf("Priority: ");
      scanf("%d", &processes[i].priority);
      processes[i].processID = i + 1;
      processes[i].turnaroundTime = 0;
      processes[i].waitingTime = 0;
   }

   priorityNonPreemptive(processes,
numOfProcesses);

   printf("\nProcess\tBurst Time\tPriority\tTurnaround
Time\tWaiting Time\n");
   for (i = 0; i < numOfProcesses; i++) {
      printf("%d\t%d\t\t%d\t\t%d\t\t%d\n",
processes[i].processID, processes[i].burstTime,
processes[i].priority,
            processes[i].turnaroundTime,
processes[i].waitingTime);
      avgTurnaroundTime +=
processes[i].turnaroundTime;
      avgWaitingTime += processes[i].waitingTime;
   }
   avgTurnaroundTime /= numOfProcesses;
   avgWaitingTime /= numOfProcesses;
   printf("\nAverage Turnaround Time: %.2f\n",
avgTurnaroundTime);
   printf("Average Waiting Time: %.2f\n",
avgWaitingTime);
return 0;
}
```