1.Scrieti un program C care primeste n fisiere ca argumente in linia de comanda. Programul va crea n thread-uri, alocate dinamic in memorie, fiecare thread va contoriza in 2 variabile globale numarul de litere mari si suma cifrelor din fisierele date ca si argument, fiecare thread va procesa un singur fisier, primit ca argument. Accesul ca variabilele globale se va face sincronizat, folosind variabile mutex.

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

pthread_mutex_t m;
pthread_mutex_t m2;

int nrLitere=0;
int suma=0;

void* worker(void* a){
  FILE* fptr;
  char c;
 fptr=fopen((char*)a,"r");
 if(fptr==NULL)
   {
    printf("err opening file");
    exit(1);
    }


 while(fscanf(fptr,"%c",&c)!=EOF){
      if(c>='A' && c<='Z'){
            pthread_mutex_lock(&m);
            nrLitere++;
            pthread_mutex_unlock(&m);
}
     if(c>='0' && c<='9'){
            pthread_mutex_lock(&m2);
            int aux=c-'0';
            suma=suma+aux;

            pthread_mutex_unlock(&m2);
}

}
 fclose(fptr);
 return 0;
}
int main(int argc,char **argv){
 pthread_t *t;

 int cate=argc-1;
 int i;
 t=malloc(cate*sizeof(pthread_t));

 pthread_mutex_init(&m,NULL);

 pthread_mutex_init(&m2,NULL);
 for( i=0;i<cate;i++)
  {

    pthread_create(&t[i],NULL,worker,argv[i+1]);
  }
  for( i=0;i<cate;i++)
  {
   pthread_join(t[i],NULL);
  }
  free(t);

  printf("suma este:%d \n",suma);
  printf("nr litere mari:%d\n", nrLitere);

   pthread_mutex_destroy(&m);

   pthread_mutex_destroy(&m2);

return 0;
}
```

2.Scrieti un program c care primeste ca arg in linia de comanda oricate perechi alcatuite dintr-un fisier si un caracter.Pentru fiecare pereche programul va crea cate un thread, fiecare thread va primi ca argumente o pereche de argumente. Programul va retine intr-un vector global frecventa fiecarei cifre din fisierele transmise threadurilor , fiecare thread va adauga in vectorul de frecventa frecventele fiecarei cifre din fisierul primit(de thread), fiecare frecventa pe pozitia corespunzatoare cifrei. Fiecare thread va nota intr un vectorglobal alocat dynamic pozitia primei aparitii a literei primate ca argument in fisierul primit ca argument .

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

int vector_frecventa[10];
int* aparitii;
pthread_mutex_t m;
```

```
typedef struct{
   char* f;
   char c;
} pereche;
pereche* pair;

void* worker(void* a){
  int index=*((int*)a);
 pereche* str=&pair[index];
  printf("%s\n",str->f);
  FILE* ptr;
 ptr=fopen(str->f,"r");
 if( ptr== NULL)
   {
     perror("unable to open file!\n");
     exit(2);
   }

   char  nr;
   int contor=-1, gasit=0;

   while(fscanf(ptr,"%c",&nr)!=EOF){
       contor++;
       if(nr>= '0' && nr<='9'){
            pthread_mutex_lock(&m);
            int aux=nr-'0';
            vector_frecventa[aux]++;
            pthread_mutex_unlock(&m);
}
       if (nr == str->c && gasit==0){
           gasit=1;
           aparitii[index]=contor;
       }

}
 free(a);
fclose(ptr);

return 0;
}

int main(int argc,char** argv){

 int nr_arg=argc-1;
 if (nr_arg%2!=0|| nr_arg==0){
     perror("Not enough arg!\n");
     exit(1);
 }
 int nr_threaduri=nr_arg/2;

 pthread_t* t;

 t=malloc(nr_threaduri*sizeof(pthread_t));

 pthread_mutex_init(&m,NULL);

 int i;

  pair=malloc(nr_threaduri*sizeof(pereche));
 aparitii=malloc(nr_threaduri*sizeof(int));

for(i=0;i<nr_threaduri;i++){

   int* n=malloc(sizeof(int));
   *n=i;
   pair[i].f=argv[2*i+1];
   pair[i].c=*(argv[2*i+2]);

   pthread_create(&t[i],NULL,worker,n);
 }
 for(i=0;i<nr_threaduri;i++){
   pthread_join(t[i],NULL);
  }
  free(pair);
  free(t);


  for(i=0;i<10;i++)
      printf("%d ",vector_frecventa[i]);
  printf("\n");
  for (i=0; i<nr_threaduri; i++)
      printf("%d ", aparitii[i]);
printf("\n");
free(aparitii);

  pthread_mutex_destroy(&m);
return 0;
}
```

3.Scrieti un program c care primeste ca argument in linia de comanda oricate perechi de fisiere si numere, fiecare fisier contine doar nr sin r este pozitiv.Pentru fiecare pereche programul va crea un thread, fiecare thread va primi ca argumente una din perechiile mentionate anterior . Fiecare thread va adauga la o variabila globala toate numerele din fisieru primit ca argument si vor stoca al n-lea nr (n fiind nr peimit ca parametru de catre thread) din fisier intr un vector global(alocat dynamic);

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

int suma=0;
int* numere;
pthread_mutex_t m;

typedef struct{
 char* f;
 int nr;
 int index;
}pair;

void* worker(void* a){
 pair* p=(pair*) a;
 int index0=p->index;

 printf("%s\n",p->f);
 FILE* ptr;
 ptr=fopen(p->f,"r");
 if(ptr==NULL)
  {
    perror("Unable to open file!\n");
    return NULL;
  }
  int nrr;
  int locatii=-1;
  while(fscanf(ptr,"%d",&nrr)!=EOF)
  {
    locatii++;

    pthread_mutex_lock(&m);
    suma+=nrr;

    pthread_mutex_unlock(&m);

    if(locatii==p->nr-'0'){
    numere[index0]=nrr;
  }
  }


fclose(ptr);
return 0;
}
int main(int argc,char** argv){

int nr_arg= argc-1;
 int nr_t=nr_arg/2;

 if(nr_arg%2!=0 || nr_arg==0)
   {
  perror("Not enough args!\n");
  exit(1);
   }

 pthread_t* t;
 t=malloc(nr_t*sizeof(pthread_t));
 int i;
pthread_mutex_init(&m,NULL);

 pair* p=malloc(nr_t*sizeof(pair));
 numere=malloc(nr_t*sizeof(int));


 for(i=0;i<nr_t;i++)
 {

  p[i].f=argv[2*i+1];
  p[i].nr=*(argv[2*i+2]);
  p[i].index=i;

 pthread_create(&t[i],NULL,worker,&p[i]);
 }

 for(i=0;i<nr_t;i++)
   {
     pthread_join(t[i],NULL);
   }

  printf("suma globala:%d\n",suma);
  for(i=0;i<nr_t;i++)
   printf("%d ",numere[i]);
  printf("\n");
  free(p);
  free(t);
  free(numere);
  pthread_mutex_destroy(&m);
 return 0;
}
```

4.Scrieti un program c care primeste ca si argumente in linia de comanda oricate perechi formate dintr-un fisier si un caracter.Pentru fiecare pereche primita program va crea cate un thread,fiecare thread va primi ca si argumente o pereche de argumente.Fiecare thread va prelucra argumentele in modul urmator:
-daca caracterul este o litera mica threadul va contoriza intr-o variabila globala literele mici din fisierul primit ca si argument
-daca caracterul este o litera mare threadul va contoriza intr-o variabila globala literele mari din fisieul primit ca si argument
-daca caracterul este o cifra threadul va adauga la o variabila globala valoarea totutor cifrelor din fisierul primit ca si argument.

```
#include <pthread.h>
#include<stdlib.h>
#include<stdio.h>
#include<ctype.h>
int nr_MARI=0;
int nr_mici=0;
pthread_mutex_t m;
int suma=0;

typedef struct{
 char* f;
 char c;
}pair;

void* worker(void* a)
  {
      pair* pa=(pair*)a;
      FILE* ptr;
      printf("fisier:%s\n",pa->f);
      ptr=fopen(pa->f,"r");
      if(ptr==NULL)
      {
        perror("unable to open file!\n");
        return NULL;
      }

      printf("litera:%c\n",pa->c);
      char c1;

      while(fscanf(ptr,"%c",&c1)!=EOF)
      {
        if(isupper(c1)&&isupper(pa->c))
        {
            pthread_mutex_lock(&m);
            nr_MARI++;
            pthread_mutex_unlock(&m);
}
```

```
          if(islower(c1)&&islower(pa->c))
          {
               pthread_mutex_lock(&m);
               nr_mici++;
               pthread_mutex_unlock(&m);
          }
         if(c1>='0' && c1<='9'&& pa->c>='0' && pa->c<='9')
         {
                pthread_mutex_lock(&m);
                suma+=c1-'0';
                pthread_mutex_unlock(&m);
         }
     }

        fclose(ptr);
        return 0;
}

int main(int argc,char** argv)
{
  int nr_arg=argc-1;
  int nr_t=nr_arg/2;

 if(nr_arg %2!=0 || nr_arg==0)
   {
     perror("Invalid nr of args!\n");
     exit(1);
   }

 pthread_t* t;
 t=malloc(nr_t*sizeof(pthread_t));

 pthread_mutex_init(&m,NULL);

 int i;
 pair* p=malloc(nr_t*sizeof(pair));
 for(i=0;i<nr_t;i++)
 {
       p[i].f=argv[2*i+1];
       p[i].c=*(argv[2*i+2]);
       pthread_create(&t[i],NULL,worker,&p[i]);
 }

 for(i=0;i<nr_t;i++)
  {
    pthread_join(t[i],NULL);
  }
 free(p);
 free(t);
 pthread_mutex_destroy(&m);
 printf("nr mari:%d\n",nr_MARI);
 printf("nr mici:%d\n",nr_mici);
 printf("suma cifrelor:%d\n",suma);
 return 0;

}
```

5.Scrieti un program c care primeste ca argumente in linia de comanda oricate perechi formate dintr un fisier si un numar.Pentru fiecare pereche programul va crea cate un thread , fiecare thread va primi ca si argumente cate o pereche.Programul va retine intr-un vector alocat dynamic numarul de litere mari din fiecare fisier, vectorul va fi populat de catre threaduri, fiecare thread va stoca pe o pozitie din vector nr de litere mari din fisierul primit ca argument.Threadurile vor contoriza intr o variabila globala cifrele aflate in primele n caractere ale fisierului .

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

int* v;
int contor=0;
pthread_mutex_t m;

typedef struct{
 char* f;
 int n;
 int index;
}pair;

void* worker(void* a)
{
        pair* pa=(pair*)a;
        FILE* ptr;
        ptr=fopen(pa->f,"r");
        if(ptr==NULL)
        {
           perror("Unable to open file!\n");
           exit(2);
        }
        char ca;
        int loc=-1;
        int cate=0;
        while(fscanf(ptr,"%c",&ca)!=EOF)
        {
          loc++;

          if(isupper(ca))
              {
                cate++;
              }
          if(ca>='0' && ca<='9'&& loc<=pa->n-'0')
              {
               pthread_mutex_lock(&m);
               contor++;
               pthread_mutex_unlock(&m);
              }
        }
        v[pa->index]=cate;

fclose(ptr);
return 0;
}
int main(int argc, char** argv)
{
        int nr_arg=argc-1;
        int nr_t=nr_arg/2;
        if(nr_arg%2!=0 || nr_arg==0)
        {
          perror("Invalid nr of args!\n");
          exit(1);
        }

        pthread_t* t;
        int i;
        t=malloc(nr_t*sizeof(pthread_t));

        pthread_mutex_init(&m,NULL);

        pair* p=malloc(nr_t*sizeof(pair));
        v=malloc(nr_t*sizeof(int));
        for(i=0;i<nr_t;i++)
        {
          p[i].f=argv[2*i+1];
          p[i].n=*(argv[2*i+2]);
          p[i].index=i;
          pthread_create(&t[i],NULL,worker,&p[i]);

        }

        for(i=0;i<nr_t;i++)
        {
          pthread_join(t[i],NULL);
        }

        pthread_mutex_destroy(&m);
        printf("contor:%d\n",contor);
        for(i=0;i<nr_t;i++)
                printf("%d ",v[i]);
        printf("\n");
        free(v);
        free(p);
        free(t);
return 0;
}
```