```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>

//Group F
//Rasheed Abid

struct Person
{
   int id;
   char name[45];
   float satisfactionLevel;
   int projects;
   int avg_hours;
   int timeCompany;
   int workAccidents;
   int promotion;
   char jobTitle[55];
   double basePay;
   double overTime;
   double benifit;
   char status[5];
   //total of 13 variables
};

struct Person2
{
```

```
    int id;
    char *name[45];
    float satisfactionLevel;
    int projects;
    int avg_hours;
    int timeCompany;
    int workAccidents;
    int promotion;
    char *jobTitle[55];
    double basePay;
    double overTime;
    double benifit;
    char *status[5];
    //total of 13 variables
};


// Global call for Person
struct Person q[15], tempq[15];
int queueCount = 0;


struct Person initialize(struct Person a) //initializes a structure
{
    a.id = -1;
    strcpy(a.name, "Void");
    a.satisfactionLevel = 0.0;
    a.projects = -1;
    a.avg_hours = 0;
```

```
    a.timeCompany = -1;
    a.workAccidents = -1;
    a.promotion = -1;
    strcpy(a.jobTitle, "");
    a.basePay = 0.0;
    a.overTime = -1.0;
    a.benifit = 0.0;
    strcpy(a.status, "TT");

    return a;
}


void printStruct(struct Person a) // prints a structure when needed
{
    printf("\nid = %d\n", a.id);
    printf("Name = %s\n", a.name);
    printf("Satisfaction Level = %f\n", a.satisfactionLevel);
    printf("Number of projects = %d\n", a.projects);
    printf("Average Hours = %d\n", a.avg_hours);
    printf("Time at company in years = %d\n", a.timeCompany);
    printf("Work Accidents = %d\n", a.workAccidents);
    printf("Promotions in last 5 years = %d\n", a.promotion);
    printf("Job Title = %s\n", a.jobTitle);
    printf("Base Pay = %lf\n", a.basePay);
    printf("Overtime done = %lf\n", a.overTime);
    printf("Benifit received = %lf\n", a.benifit);
    printf("Status of Job = %s\n", a.status);
}
```

```
void AssistantQueryInfo(struct Person a) // prints assistant query when needed
{
   printf("Assistant is asking the history file with the following inforamtion:\n");
   //printf("\nid = %d\n", a.id);
   printf("\nName = %s\n", a.name);
   //printf("Satisfaction Level = %f\n", a.satisfactionLevel);
   //printf("Number of projects = %d\n", a.projects);
   //printf("Average Hours = %d\n", a.avg_hours);
   //printf("Time at company in years = %d\n", a.timeCompany);
   //printf("Work Accidents = %d\n", a.workAccidents);
   //printf("Promotions in last 5 years = %d\n", a.promotion);
   printf("Job Title = %s\n", a.jobTitle);
   //printf("Base Pay = %lf\n", a.basePay);
   //printf("Overtime done = %lf\n", a.overTime);
   //printf("Benifit received = %lf\n", a.benifit);
   printf("Status of Job = %s\n", a.status);
}


struct Person HistoryFun(int CallId, struct Person a)
{

   printf("Current queue size = %d\n\n", queueCount);
   q[13] = initialize(q[13]);
   if(CallId == 0)
   {
      if(queueCount >= 10) //allow the queue to work around
      {
```

```
        FILE *fp;
        fp = fopen("History.txt", "w");
        for(int i=0; i<queueCount; i++) // copy to temp
        {
           tempq[i] = q[i];
        }

        int j = 10; //add the incoming query to the end of the temp queue
        tempq[j] = a;

        //print the queue in file here


        for(int i=0; i<queueCount; i++) // copy temp to the original queue
        {
           q[i] = tempq[i+1];
           struct Person aa = q[i];
           fprintf(fp, "%s\n%s\n%s\n %d %f %d %d %d %d %lf %lf %lf\n\n", aa.name,a.jobTitle,
aa.status, aa.id, aa.satisfactionLevel, aa.projects, aa.avg_hours, aa.timeCompany,
aa.workAccidents, aa.promotion, aa.basePay,aa.overTime, aa.benifit);
        }
        fclose(fp);
        strcpy( q[13].name,"Completed" );
    }

    else
    {
        FILE *fp;
        fp = fopen("History.txt", "a");
```

```
        q[queueCount] = a;
        struct Person aa = q[queueCount];
        fprintf(fp, "%s\n%s\n%s\n %d %f %d %d %d %d %lf %lf %lf\n\n", aa.name,a.jobTitle,
aa.status, aa.id, aa.satisfactionLevel, aa.projects, aa.avg_hours, aa.timeCompany,
aa.workAccidents, aa.promotion, aa.basePay,aa.overTime, aa.benifit);
        strcpy( q[13].name,"Completed" );

        queueCount++;

        fclose(fp);
      }
   }

   else //When the assistant calls in for query
   {

      for(int i = 0; i<queueCount ; i++)
      {
         if( strcmp(q[i].name, a.name) == 0)
         {
            if(strcmp(q[i].jobTitle, a.jobTitle) == 0 && strcmp(q[i].status, a.status) == 0)
            {
               return q[i];
            }
         }
      }
   }
   return q[13];
}
```

```
void* historyFunction(void* received_struct)
{
   struct Person2 *requestedEmployee = (struct Person2*) received_struct;
   struct Person a;
   int CallId = 1;
   //printf("Current queue size = %d\n\n", queueCount);
   q[13] = initialize(q[13]);
   if(CallId == 0)
   {
      if(queueCount >= 10) //allow the queue to work around
      {
         FILE *fp;
         fp = fopen("History.txt", "w");
         for(int i=0; i<queueCount; i++) // copy to temp
         {
            tempq[i] = q[i];
         }

         int j = 10; //add the incoming query to the end of the temp queue
         tempq[j] = a;

         //print the queue in file here


         for(int i=0; i<queueCount; i++) // copy temp to the original queue
         {
            q[i] = tempq[i+1];
            struct Person aa = q[i];
```

```
            fprintf(fp, "%s\n%s\n%s\n %d %f %d %d %d %d %lf %lf %lf\n\n", aa.name,a.jobTitle,
aa.status, aa.id, aa.satisfactionLevel, aa.projects, aa.avg_hours, aa.timeCompany,
aa.workAccidents, aa.promotion, aa.basePay,aa.overTime, aa.benifit);
          }
          fclose(fp);
          strcpy( q[13].name,"Completed" );
      }

      else
      {
          FILE *fp;
          fp = fopen("History.txt", "a");

          q[queueCount] = a;
          struct Person aa = q[queueCount];
          fprintf(fp, "%s\n%s\n%s\n %d %f %d %d %d %d %lf %lf %lf\n\n", aa.name,a.jobTitle,
aa.status, aa.id, aa.satisfactionLevel, aa.projects, aa.avg_hours, aa.timeCompany,
aa.workAccidents, aa.promotion, aa.basePay,aa.overTime, aa.benifit);
          strcpy( q[13].name,"Completed" );

          queueCount++;

          fclose(fp);
      }
   }

   else //When the assistant calls in for query
   {

      for(int i = 0; i<queueCount ; i++)
```

```
        {
            if( strcmp(q[i].name, a.name) == 0)
            {
                if(strcmp(q[i].jobTitle, a.jobTitle) == 0 && strcmp(q[i].status, a.status) == 0)
                {
                    //return q[i];
                }
            }
        }
    }
    //return q[13];
    return(void*)requestedEmployee;
}

/*
int main()
{
    struct Person Assistant;
    Assistant = initialize(Assistant);


    for(int ask = 1; ask < 5; ask++)
    {

        printf("\n\nQuery number = %d\n\n", ask);
        Assistant = AssistantQuery( (ask%7) % 2, Assistant);
        //AssistantQueryInfo(Assistant);
```

```
      //Make the assistant query #ask to the HistoryFun()
      struct Person SearchResults;
      SearchResults = HistoryFun(1, Assistant);
      if( strcmp(SearchResults.name, "Void") != 0 ) //the name is not "Void"
      {
         //The search has been successful
         //Print the result from the assistant to display
         //For simplicity, we are just printing the result here.
         printf("Searching was successful. The data was found on the history files....\n");
         printStruct(SearchResults);
      }
      else
      {
         //The search result has failed
         printf("The Search was UNSUCCUSSFUL. The system is calling back the server and will
retrieve the information soon.....\n");

         SearchResults = Server(Assistant);
         SearchResults = HistoryFun(0, SearchResults);
         //printStruct(SearchResults);

         SearchResults = HistoryFun(1, Assistant);
         printStruct(SearchResults);
      }
   }
   return 0;
}
*/
```