```c
//
//  main.c
//  assign9
//
//  Created by Ayoola Nurudeen Etiko on 2020-11-29.
//  Disk scheduling Algorithms
//  FCFS    (F)
//  SSTF    (T)
//  C-SCAN  (C)
//  LOOK    (L)
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <time.h>
#define REQUESTS 10000

void * fcfs(int * req, int *tmereq, int * check, int head, const char * direction);
void * sstf(int * req, int *tmereq, int * check, int head, const char * direction);
void * cscan(int * req, int *tmereq, int * check, int head, const char * direction);
void * look(int * req, int *tmereq, int * check, int head, const char * direction);
void * printOutput(int headmvm,int ttime);

int main(int argc, const char * argv[]) {
    int requests[REQUESTS];
    int timereq[REQUESTS];
    int checked[REQUESTS];
    int i=0, j;
    int sec = -1;
    int tme = -1;
    int head = atoi(argv[2]);
    const char * direction = argv[3];

    //    printf("head-> %d direction -> %s\n", head, direction);

    for(j=0; j<REQUESTS; j++){
        requests[j] = -1;
        timereq[j] = -1;
        checked[j] = -1;
    }

    while(scanf("%d %d", &sec, &tme) == 2){
        requests[i] = sec;
        timereq[i] = tme;
        checked[i] = 0;
        //        printf("\nlocation: %d time: %d", sec, tme);
```

```
        i++;
    }if(strcmp(argv[1], "F")==0){
        fcfs(requests, timereq, checked, head, direction);
    }else if(strcmp(argv[1],"T")==0){
        sstf(requests, timereq, checked, head, direction);
    }else if(strcmp(argv[1],"C")==0){
        cscan(requests, timereq, checked, head, direction);
    }else if(strcmp(argv[1],"L")==0){
        look(requests, timereq, checked, head, direction);
    }


    return 0;
}

void * fcfs(int * req, int *tmereq, int * check, int head, const char * direction){
    int locTime = 0;
    int k=0;
    int headmvm = 0;
    int reversed = 0;
    int ttime = 0;
    int dir = 0;
    int temp = 0;
    int s = 0;

    for(s=0;s<REQUESTS;s++){
        if(req[s] == -1){
            break;
        }
    }
    while((k<REQUESTS && req[k] != -1)||temp != 0){
        if(temp > 0){
            temp--;
            locTime++;
        }else{
            if(tmereq[k] <= locTime){
                //printf("\nWere in here");
                temp = abs((head - req[k])) / 10;
                //direction check
                if(k == 0){
                    if(req[k] > head && strcmp(direction, "a") == 0){
                        //forward
                        dir = 1;
                    }if(strcmp(direction, "d") == 0){
                        //backwards
                        dir = -1;
                    }
```

```
                }if(req[k] > head && strcmp(direction, "d") == 0){
                    reversed++;
                    dir = 1;
                    temp+=5;
                }if(req[k] < head && strcmp(direction, "a") == 0){
                    reversed++;
                    dir = -1;
                    temp+=5;
                }
                head = req[k];
                headmvm++;
                ttime += temp;
                k++;
            }
            locTime++;
        }
    }
    printOutput(headmvm,ttime);
    return 0;
}

void * sstf(int * req, int *tmereq, int * check, int head, const char * direction){
    int locTime = 0;
    int k=0;
    int headmvm = 0;
    int reversed = 0;
    int ttime = 0;
    int dir = 0;
    int temp = 0;
    // we need to sort the list so we need to find the end of the list
    int s = 0;
    for(s=0;s<REQUESTS;s++){
        if(req[s] == -1){
            break;
        }
    }
    int u;
    int b;
    for(u = 0; u<s; u++){
        for(b = 0; b<u; b++){
            //            printf("b->%d u->%d\n", req[b], req[u]);
            if(abs(head - req[b]) > abs(head - req[u])){
                int tempa = req[u];
                req[u] = req[b];
                req[b] = tempa;
                tempa = tmereq[u];
                tmereq[u] = tmereq[b];
                tmereq[b] = tempa;
```

```
            }
        }
    }

    while(k<s||temp != 0){
        if(temp > 0){
            temp--;
            locTime++;
        }else{
            if(tmereq[k] <= locTime){
                temp = abs((head - req[k])) / 10;
                //direction check
                if(k == 0){
                    if(req[k] > head && strcmp(direction, "a") == 0){
                        //forward
                        dir = 1;
                    }if(strcmp(direction, "d") == 0){
                        //backwards
                        dir = -1;
                    }
                }if(req[k] > head && strcmp(direction, "d") == 0){
                    reversed++;
                    dir = 1;
                    temp+=5;
                }if(req[k] < head && strcmp(direction, "a") == 0){
                    reversed++;
                    dir = -1;
                    temp+=5;
                }
                head = req[k];
                headmvm++;
                ttime += temp;
                k++;
            }
            locTime++;
        }
    }
    printOutput(headmvm, ttime);
    return 0;
}

void * cscan(int * req, int *tmereq, int * check, int head, const char * direction){
    int locTime = 0;
    int k=0;
    int headmvm = 0;
    int reversed = 0;
    int ttime = 0;
    int dir = 0;
```

```
    int temp = 0;
    int pivot = 0;
    // we need to sort the list so we need to find the end of the list
    int s = 0;
    for(s=0;s<REQUESTS;s++){
        if(req[s] == -1){
            req[s] = head;
            break;
        }
    }
    // we need to sort this into 2 blocks; after head and before head.
    int u;
    int b;
    for(u = 0; u<=s; u++){
        for(b = 0; b<u; b++){
            if(abs(head - req[b]) > abs(head - req[u])){
                int tempa = req[u];
                req[u] = req[b];
                req[b] = tempa;
                tempa = tmereq[u];
                tmereq[u] = tmereq[b];
                tmereq[b] = tempa;
                if(tmereq[b] == -1){
                    pivot = b;
                }
            }
        }
    }
    // pt 1;
    k = pivot+1;
    while(k<=s||temp != 0){
        if(temp > 0){
            temp--;
            locTime++;
        }else{
            if(tmereq[k] <= locTime){
                temp = abs((head - req[k])) / 10;
                //direction check
                if(k == 0){
                    if(req[k] > head && strcmp(direction, "a") == 0){
                        //forward
                        dir = 1;
                    }if(strcmp(direction, "d") == 0){
                        //backwards
                        dir = -1;
                    }
                }if(req[k] > head && strcmp(direction, "d") == 0){
                    reversed++;
```

```
                dir = 1;
                temp+=5;
            }if(req[k] < head && strcmp(direction, "a") == 0){
                reversed++;
                dir = -1;
                temp+=5;
            }
            headmvm++;
            ttime += temp;
            k++;
        }
        locTime++;
    }
}
//pt 2
k=0;
while(k<pivot||temp != 0){
    if(temp > 0){
        temp--;
        locTime++;
    }else{
        if(tmereq[k] <= locTime){
            temp = abs((head - req[k])) / 10;
            //direction check
            if(k == 0){
                if(req[k] > head && strcmp(direction, "a") == 0){
                    //forward
                    dir = 1;
                }if(strcmp(direction, "d") == 0){
                    //backwards
                    dir = -1;
                }
            }if(req[k] > head && strcmp(direction, "d") == 0){
                reversed++;
                dir = 1;
                temp+=5;
            }if(req[k] < head && strcmp(direction, "a") == 0){
                reversed++;
                dir = -1;
                temp+=5;
            }
            headmvm++;
            ttime += temp;
            k++;
        }
        locTime++;
    }
}
```

```
    printOutput(headmvm, ttime);
    return 0;
}

void * look(int * req, int *tmereq, int * check, int head, const char * direction){
    int locTime = 0;
    int k=0;
    int headmvm = 0;
    int ttime = 0;
    int dir = 0;
    int temp = 0;
    int s = 0;
    for(s=0;s<REQUESTS;s++){
        if(req[s] == -1){
            break;
        }
    }
    int u;
    int max = req[0];
    int min = req[0];
    for(u = 0; u<s; u++){
        if(max < req[u]){
            max = req[u];
        }if(min > req[u]){
            min = req[u];
        }
    }

    while(k<1){
        if(temp > 0){
            temp--;
            locTime++;
        }else{
            if(tmereq[k] <= locTime){

                //direction check
                if(k == 0){
                    if(strcmp(direction, "a")==0){
                        dir = 1;
                    }if(strcmp(direction, "d") == 0){
                        dir = -1;
                    }
                    if(dir == 1){
                        temp += abs((head - max)) / 10;
                    }
                    dir = -1;
                    head = max;
                    if(dir == -1){
```

```
                        temp += abs((head - min)) / 10;
                    }
                    dir = 1;
                    head = min;
                }
                temp +=5;
                headmvm++;
                ttime += temp;
                k++;
            }
            locTime++;
        }
    }
    printOutput(headmvm, ttime);
    return 0;
}

void * printOutput(int headmvm, int ttime){
    printf("\nTotal amount of head movements required: %d \n", headmvm);
    printf("Total time required to service all requests: %d \n", ttime);
    return 0;
}
```