c

Car on Campus Project

Author:蒋腾飞 张雯琪

Date: 2020-01-02

Chapter 1: Introduction

1.1 Problem description:

Zhejiang University has 8 campuses, each with several gates. This time we gonna record every car crossing the gate by its plate number and the in/out time. For any appointed time, we should count the number of cars parking on campus then and at last find the car with the longest parking time during the day. If there are more than one car with the same longest time, then print then in an alphabetical order.

1.2 Algorithm Background

1.2.1 Quick sort Algorithm

qsort is a C standard library function that implements a polymorphic sorting algorithm for arrays of arbitrary objects according to a user-provided comparison function.

It's function prototype:

void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))

base —— The pointer to the first element of the array to be sorted.

nitems —— The number of elements in the array pointed to by base.

size —— The size of each element in the array, in bytes.

compar —— A function used to compare two elements, a function pointer

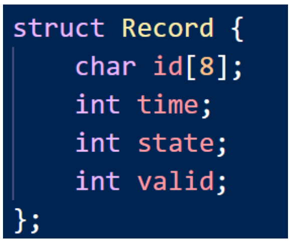
1.2.2 time.h

To use C's standard library time.h to measure the performance of the function. By repeating the function calls for K times, we can obtain a long enough total run time and divide it to get a more accurate duration for a single run of the function.

Chapter 2: Algorithm Specification

2.1 Main data structures

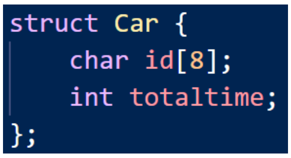
2.1.1 Record



```c
struct Record {
    char id[8];
    int time;
    int state;
    int valid;
};
```

For each record, store the plate name of the car in ‘Record.id[]’, store the given time point in term of seconds in ‘Record.time’, define ‘Record.state’ to record whether the car is in or out of the campus, and define ‘Record.valid’, initially zero, to record whether its state is valid or not.

2.1.2 Car



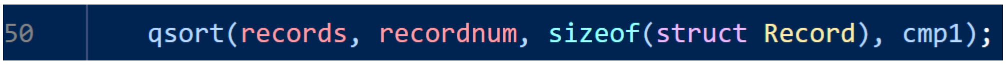
```c
struct Car {
    char id[8];
    int totaltime;
};
```

For all the identical car, store its plate name in ‘Car.id[]’, and store its total parking time on campus in ‘Car.totaltime’.

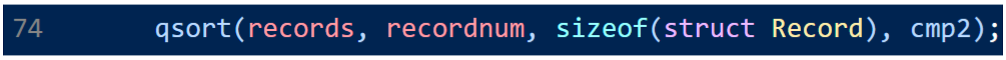
2.2 Algorithms

2.2.1 Quick sort

Call C standard library function qsort()



```c
    qsort(records, recordnum, sizeof(struct Record), cmp1);
```

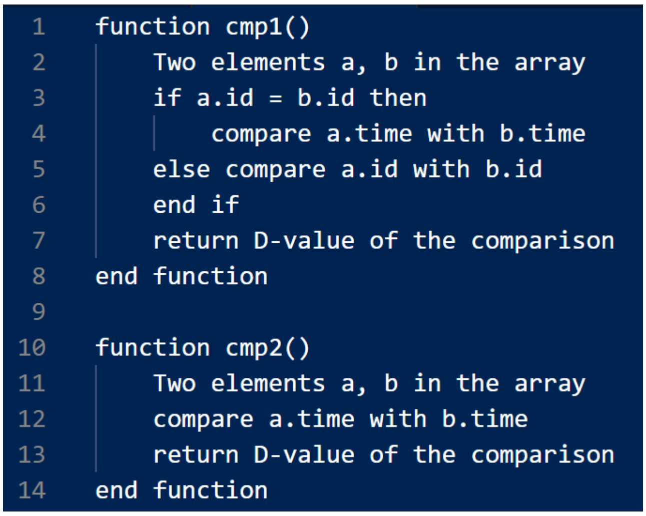


```c
    qsort(records, recordnum, sizeof(struct Record), cmp2);
```

‘records’ is the pointer to the first element of the array to be sorted, ‘recordnum’ is the number of elements in the array, and ‘cmp1’‘cmp2’ are function pointers, pseudocode showed below.

The first qsort completes the sort process by the car’s plate name, if the two has the same name then sort by the in/out time.

The second qsort completes the sort process simply by the in/out time.



```
function cmp1()
    Two elements a, b in the array
    if a.id = b.id then
        compare a.time with b.time
    else compare a.id with b.id
    end if
    return D-value of the comparison
end function

function cmp2()
    Two elements a, b in the array
    compare a.time with b.time
    return D-value of the comparison
end function
```

2.2.2 Main function

Input recordnum, querynum and *records

Sort *records by the plate name of each cars in a alphabetical order, those with same name are sorted by the in/out time

For two adjacent elements in *records, if they are same then judge the validity and update the total time if positive. If they are different then update the number of cars. This way each valid record can be stored in *cars.

Sort *records again simly by the in/out time.

Input *querytime

For each query time, find all the records that are valid and earlier than the querytime, plus or minus one to the number of cars on campus(coc) according to its state, which is whether in or out. Then print out the number of cars for each query time.

Find out the maximum number of the total parking time

Find out all the cars with the maximum parking time and print them out in a alphabetical order.

Print out the maximum parking time in term of hour:minute:second

Chapter 3: Testing Results

Case 1(from PTA, for typical test cases):

	Purpose	Expected Result	Actual Result	Runtime /ms	RAM /kb
1	Invalid input and same maximum time	AC/AC	AC/AC	4	256
2	The first in and the last out car is the same, one car is in/out in the given query time	AC/AC	AC/AC	20	640
3	N cars in before T and out after T	AC/AC	AC/AC	20	700
4	Smallest N	AC/AC	AC/AC	4	256
5	Maximum N	AC/AC	AC/AC	62	956
6	Maximum parallel number	AC/AC	AC/AC	19	672

Other cases:

In chapter 4 we write a function to randomly create input data to test the time complexity, thus omitted here.

Chapter 4: Analysis and Comments

4.1 Time complexity analysis

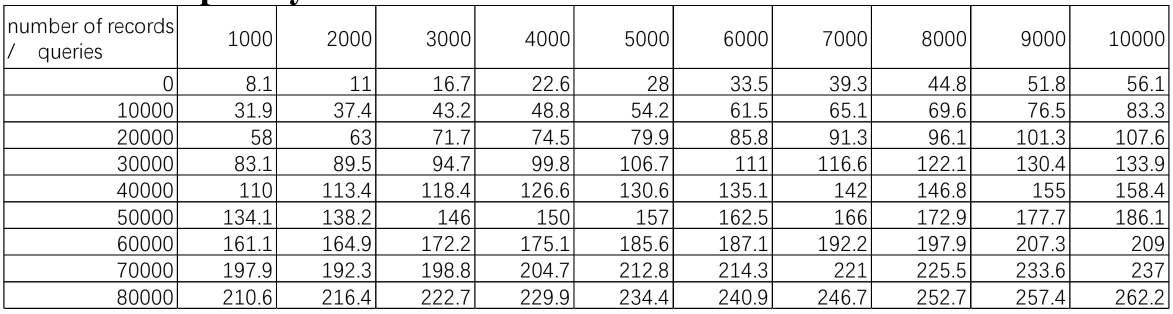
The time complexity is determined by two variables, ‘recordnum’, represented by Nr, and ‘querynum’, represented by Nq.

For qsort, its time complexity is O(NrlogNr)

In main function, there are 7 loops, 5 of them have O(Nr) as their time complexity, 1 of them has O(Nq) and the other has O(Nr*Nq)

Thus the time complexity is O(NrlogNr+Nr*Nq)

4.2 Time complexity test



number of records / queries	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
0	8.1	11	16.7	22.6	28	33.5	39.3	44.8	51.8	56.1
10000	31.9	37.4	43.2	48.8	54.2	61.5	65.1	69.6	76.5	83.3
20000	58	63	71.7	74.5	79.9	85.8	91.3	96.1	101.3	107.6
30000	83.1	89.5	94.7	99.8	106.7	111	116.6	122.1	130.4	133.9
40000	110	113.4	118.4	126.6	130.6	135.1	142	146.8	155	158.4
50000	134.1	138.2	146	150	157	162.5	166	172.9	177.7	186.1
60000	161.1	164.9	172.2	175.1	185.6	187.1	192.2	197.9	207.3	209
70000	197.9	192.3	198.8	204.7	212.8	214.3	221	225.5	233.6	237
80000	210.6	216.4	222.7	229.9	234.4	240.9	246.7	252.7	257.4	262.2

4.3 Space complexity analysis

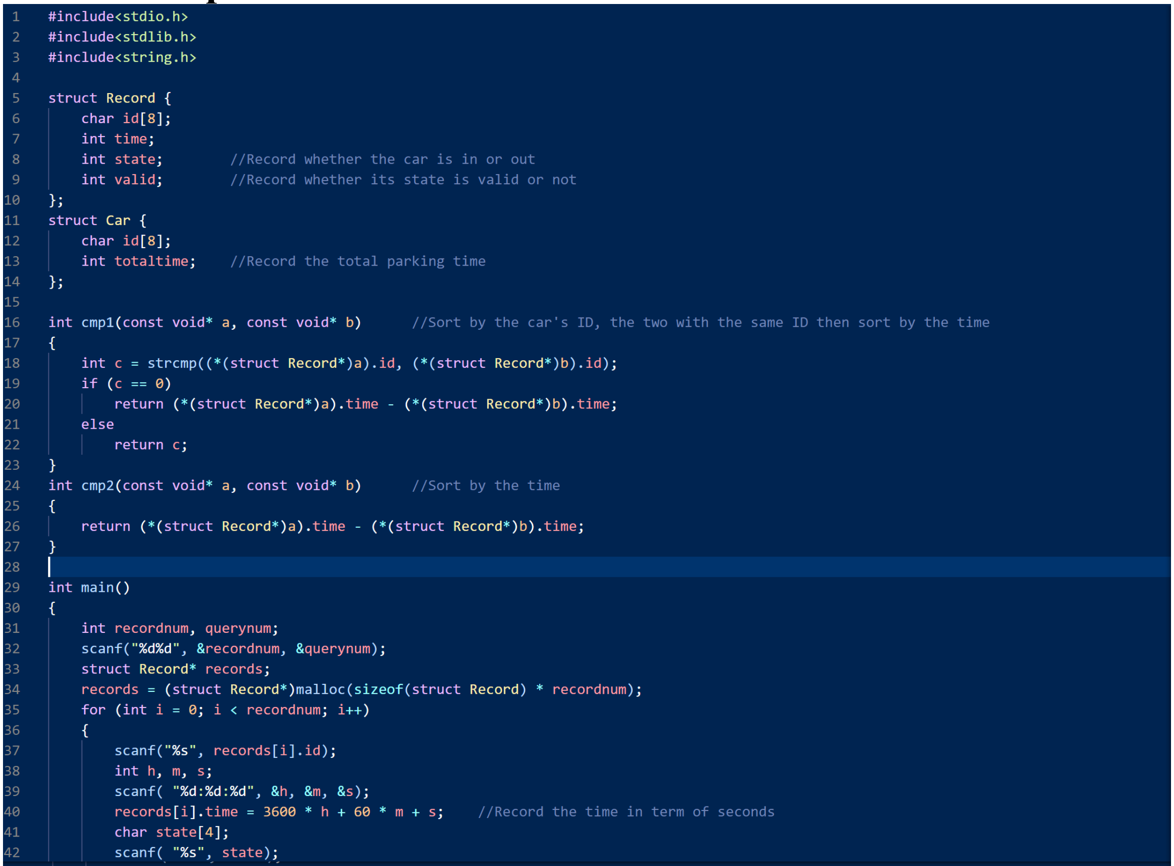
The space complexity is also determined by two variables, ‘recordnum’, represented by Nr, and ‘querynum’, represented by Nq.

There are three main data structures, *records for O(Nr), *cars for O(Nr) and *querytime for O(Nq)

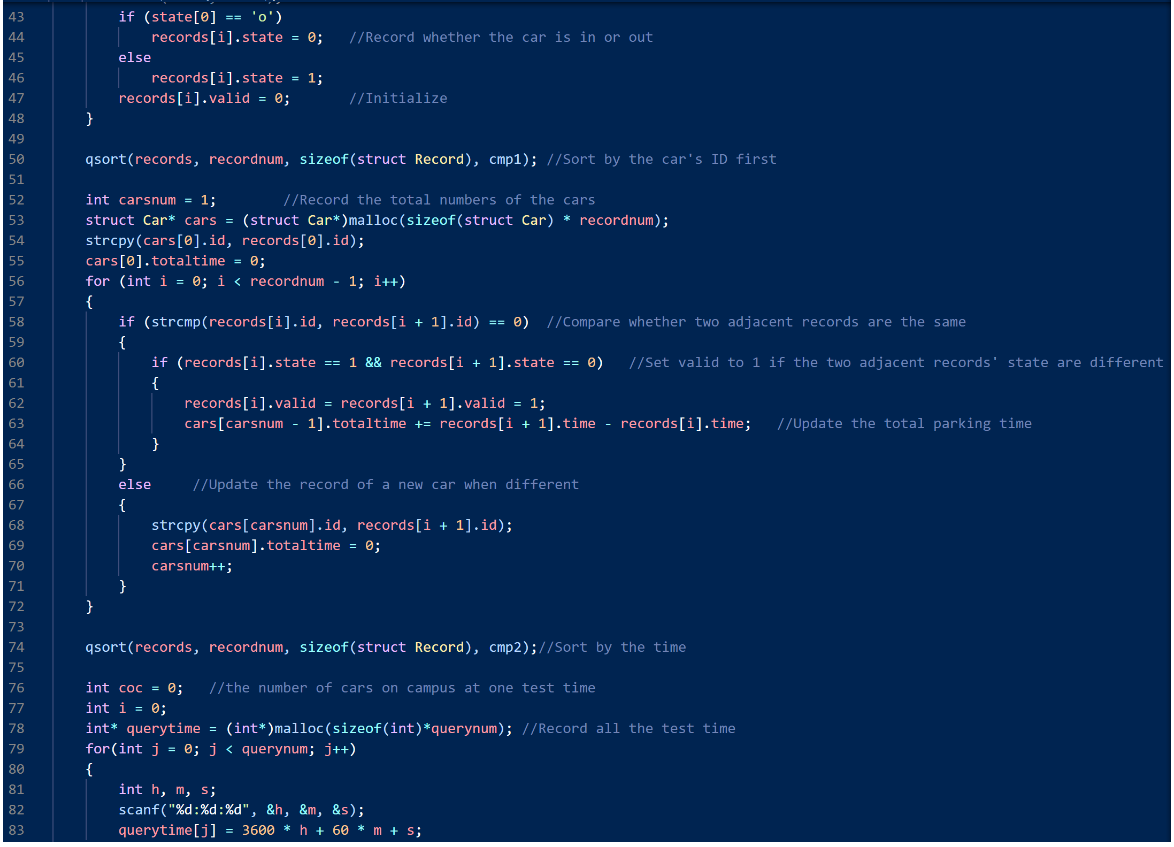
Thus the space complexity is O(Nr+Nq)

Appendix: Source Code (in C)

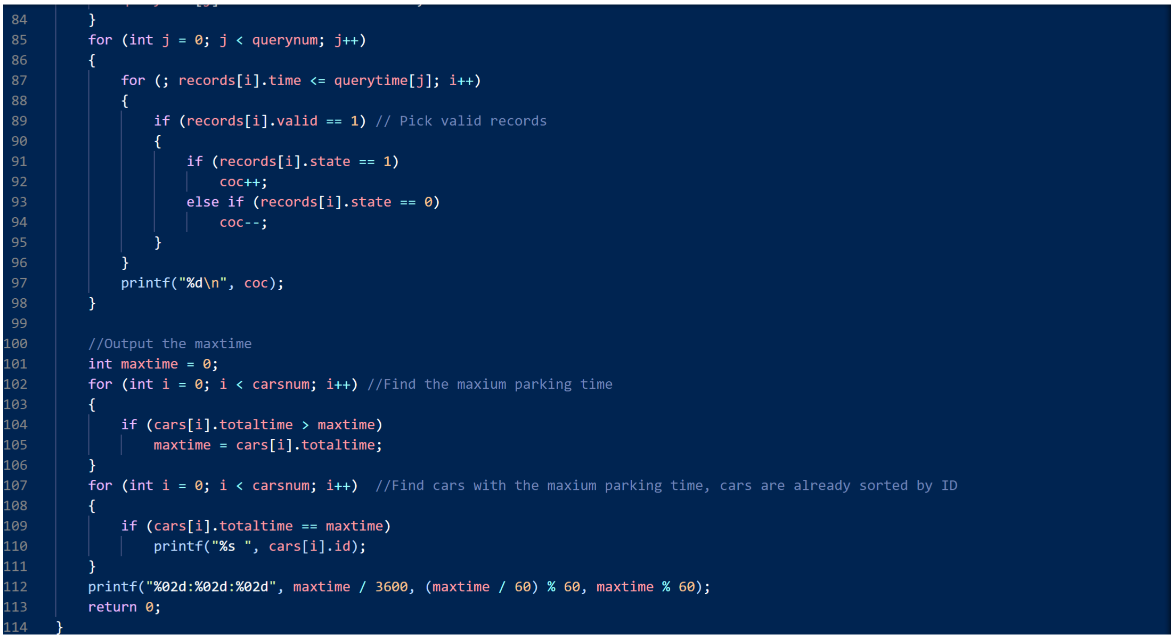
CarOnCampus.c



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

struct Record {
    char id[8];
    int time;
    int state;        //Record whether the car is in or out
    int valid;        //Record whether its state is valid or not
};
struct Car {
    char id[8];
    int totaltime;    //Record the total parking time
};

int cmp1(const void* a, const void* b)      //Sort by the car's ID, the two with the same ID then sort by the time
{
    int c = strcmp((*(struct Record*)a).id, (*(struct Record*)b).id);
    if (c == 0)
        return (*(struct Record*)a).time - (*(struct Record*)b).time;
    else
        return c;
}
int cmp2(const void* a, const void* b)      //Sort by the time
{
    return (*(struct Record*)a).time - (*(struct Record*)b).time;
}

int main()
{
    int recordnum, querynum;
    scanf("%d%d", &recordnum, &querynum);
    struct Record* records;
    records = (struct Record*)malloc(sizeof(struct Record) * recordnum);
    for (int i = 0; i < recordnum; i++)
    {
        scanf("%s", records[i].id);
        int h, m, s;
        scanf( "%d:%d:%d", &h, &m, &s);
        records[i].time = 3600 * h + 60 * m + s;    //Record the time in term of seconds
        char state[4];
        scanf( "%s", state);
```

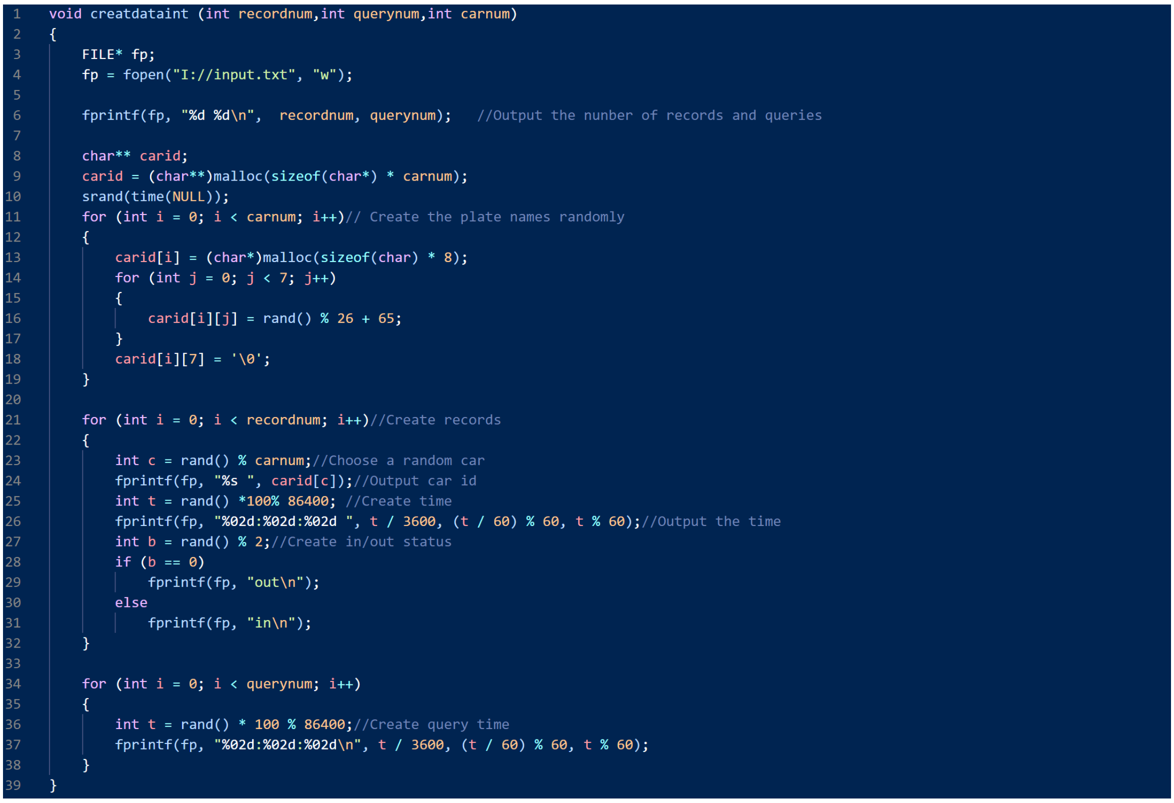


```
        if (state[0] == 'o')
            records[i].state = 0;    //Record whether the car is in or out
        else
            records[i].state = 1;
        records[i].valid = 0;        //Initialize
    }

    qsort(records, recordnum, sizeof(struct Record), cmp1); //Sort by the car's ID first

    int carsnum = 1;        //Record the total numbers of the cars
    struct Car* cars = (struct Car*)malloc(sizeof(struct Car) * recordnum);
    strcpy(cars[0].id, records[0].id);
    cars[0].totaltime = 0;
    for (int i = 0; i < recordnum - 1; i++)
    {
        if (strcmp(records[i].id, records[i + 1].id) == 0)  //Compare whether two adjacent records are the same
        {
            if (records[i].state == 1 && records[i + 1].state == 0)   //Set valid to 1 if the two adjacent records' state are different
            {
                records[i].valid = records[i + 1].valid = 1;
                cars[carsnum - 1].totaltime += records[i + 1].time - records[i].time;   //Update the total parking time
            }
        }
        else     //Update the record of a new car when different
        {
            strcpy(cars[carsnum].id, records[i + 1].id);
            cars[carsnum].totaltime = 0;
            carsnum++;
        }
    }

    qsort(records, recordnum, sizeof(struct Record), cmp2);//Sort by the time

    int coc = 0;    //the number of cars on campus at one test time
    int i = 0;
    int* querytime = (int*)malloc(sizeof(int)*querynum); //Record all the test time
    for(int j = 0; j < querynum; j++)
    {
        int h, m, s;
        scanf("%d:%d:%d", &h, &m, &s);
        querytime[j] = 3600 * h + 60 * m + s;
```



```c
    }
    for (int j = 0; j < querynum; j++)
    {
        for (; records[i].time <= querytime[j]; i++)
        {
            if (records[i].valid == 1) // Pick valid records
            {
                if (records[i].state == 1)
                    coc++;
                else if (records[i].state == 0)
                    coc--;
            }
        }
        printf("%d\n", coc);
    }

    //Output the maxtime
    int maxtime = 0;
    for (int i = 0; i < carsnum; i++) //Find the maxium parking time
    {
        if (cars[i].totaltime > maxtime)
            maxtime = cars[i].totaltime;
    }
    for (int i = 0; i < carsnum; i++)  //Find cars with the maxium parking time, cars are already sorted by ID
    {
        if (cars[i].totaltime == maxtime)
            printf("%s ", cars[i].id);
    }
    printf("%02d:%02d:%02d", maxtime / 3600, (maxtime / 60) % 60, maxtime % 60);
    return 0;
}
```

CreateData.c



```c
void creatdataint (int recordnum,int querynum,int carnum)
{
    FILE* fp;
    fp = fopen("I://input.txt", "w");

    fprintf(fp, "%d %d\n",  recordnum, querynum);   //Output the nunber of records and queries

    char** carid;
    carid = (char**)malloc(sizeof(char*) * carnum);
    srand(time(NULL));
    for (int i = 0; i < carnum; i++)// Create the plate names randomly
    {
        carid[i] = (char*)malloc(sizeof(char) * 8);
        for (int j = 0; j < 7; j++)
        {
            carid[i][j] = rand() % 26 + 65;
        }
        carid[i][7] = '\0';
    }

    for (int i = 0; i < recordnum; i++)//Create records
    {
        int c = rand() % carnum;//Choose a random car
        fprintf(fp, "%s ", carid[c]);//Output car id
        int t = rand() *100% 86400; //Create time
        fprintf(fp, "%02d:%02d:%02d ", t / 3600, (t / 60) % 60, t % 60);//Output the time
        int b = rand() % 2;//Create in/out status
        if (b == 0)
            fprintf(fp, "out\n");
        else
            fprintf(fp, "in\n");
    }

    for (int i = 0; i < querynum; i++)
    {
        int t = rand() * 100 % 86400;//Create query time
        fprintf(fp, "%02d:%02d:%02d\n", t / 3600, (t / 60) % 60, t % 60);
    }
}
```

Declaration

We hereby declare that all the work done in this project titled "Report" is of our independent effort as a group.

Programmer: *蒋腾飞, also complete the CreateData.c*

Tester&Reporter: *张雯琪, also complete the whole report*