Name : Aditi Paretkar

Roll : 1911034

Experiment 7 : Priority Queue Code , Algorithm and Screenshots of Output

**Algorithm used for the program :**

My program will ask for the number of queries , where each query , will be either 'ENTER' or 'SERVE'. When any element is served , it will print that element which was served . At the end of the queries , it will print those elements (students) remaining to be served.

These are the steps that I followed.

1. Initialize a struct student , having a char array to store the name , a float value to store the cgpa , and an int value to store the id (The id will be used to check how many users have a same cgpa, it will be auto generated. Initially it will be 1 , for all , then suppose we have 2 students with cgpa 5 , their id's will be 1 , 2. If we have 3 students with cgpa 6, their id's will be 1,2,3 , depending on the order of who came first. The student who was entered first will have the lowest id and will be served first).

2. We have a struct prque which will contain the front and rear elements as well as a structure of student elements array.

3. we have a void initprque() function , which will set the names and id's and cgpa of every element to 0 , and will be first called by main ().

4. After that , user will enter the number of queries (n) and each query will be an 'ENTER' or 'SERVE' operation.

5. To perform ENTER , we have a function void enter(struct prque \*pq, char name[10], float cgpa,int id),initially , it will check whether queue is empty or full and initialize the front and rear pointers accordingly.

6. Now when we pass the data of any student to the function , it will first check  how many students who are already in the queue have the same cgpa as the element to be inserted. This value is stored in the 'count' variable. And the index of the last element having the same cgpa as that of the element to be inserted will be stored in the 'index' variable.

7. In case count =0, no element already in the queue has the same id as the element to be inserted. Hence we will scan the queue up to that element which has a smaller cgpa (lesser priority) than the element to be inserted , and from that element onwards , all elements will be shifted one place to the right in order to make space for the new element , and the element is inserted

8. if count !=0 , means there is at least 1 element having the same cgpa as the element to be inserted . The 'index' variable will hold the index of the last element as the one to be inserted , and then we will insert the element at the location of [index+1], by shifting all other elements one place to the right.

9. Thus we are entering elements in the queue such that the highest priority element will always be at the first.

10. In the function void serve(struct prque\*pq) , we will remove the element at index [0], and shift all the other elements one place forward. The serve function will also display the element that was served .

11. Finally at the end of n queries , we call the display function to display the list of elements remaining to be served.

**Code of the Program :**

#include <stdio.h>

#include<string.h>

#define MAX 10;

struct student

{

char n[10];

float cgpa;

int id;

};

struct prque

{

int front;

int rear;

struct student arr[10] ;

};

void initprque(struct prque\*r)

{

r->front = -1; r->rear =-1;

int i;

for(i=0;i<10;i++)

{

strcpy(r->arr[i].n,"");

r->arr[i].cgpa=0;

r->arr[i].id=0;

}

}

void enter(struct prque \*pq, char name[10], float cgpa,int id)

{

struct student temp,temp1;

int i,j,count=0,index=0;

struct student \*q = &temp;

strcpy(temp.n,name);

temp.cgpa = cgpa;

temp.id=id;

if(pq->rear==9)

{

printf("QUEUE IS FULL");

}

else if((pq->rear==-1)&&(pq->front==-1))

{

pq->rear = pq->front=0;

}

else

{

pq->rear++;

}

for(i=0;i<=pq->rear;i++)

{

if(pq->arr[i].cgpa==temp.cgpa)

{count++;/\*count variable will be used to check how many students have the same cgpa\*/

temp.id++;

index = i;/\*index stores the index of the last array element having the same cgpa as the temp variable\*/

}

}

if(count==0)/\*this will occur if no element that already exists in the queue has the same priority as the element to be inserted\*/

{

for(i=0;i<=pq->rear;i++)/\*for inserting element in decreasing order of cgpa/priority\*/

{

if(pq->arr[i].cgpa<temp.cgpa)

{

for(j=pq->rear;j>=i;j--)

{

pq->arr[j+1] = pq->arr[j];

}

pq->arr[i]=temp;

break;

}

}

}

else if(count!=0)/\*this will occur if atleast 1 element that already exists in the queue has the same priority as the element to be inserted\*/

{

for(i=pq->rear;i>index;i--)

{

pq->arr[i+1] = pq->arr[i];

}

pq->arr[index+1]=temp;

}

}

void serve(struct prque\*pq)

{

int i;

printf("Element served:\n");

printf("name: ");

puts(pq->arr[0].n);

printf(" cgpa: ");

printf("%f",pq->arr[0].cgpa);

printf(" id: ");

printf("%d\n",pq->arr[0].id);

for(i=0;i<=pq->rear;i++)

{

pq->arr[i]= pq->arr[i+1];

}/\*note-we will be shifting all array elements one place forward instead of incrementing the front pointer , because if we increment the front pointer only , that will lead to wastage of space when the front elements are deleted\*/

pq->rear--;

}

void display(struct prque\*p)

{

int i;

printf("The students remaining in the queue are as follows\n");

for(i=0;i<=p->rear;i++)

{

printf("ID: %d\n",p->arr[i].id);

printf("Name:");

puts(p->arr[i].n);

printf("CGPA:%f\n",p->arr[i].cgpa);

printf("--------------------\n");

}

}

int main(void) {

struct prque p;

initprque(&p);

char n1[10];

int ch, id1,n,i;

float cp;

printf("Enter the number of queries\n");

scanf("%d",&n);

printf("press 1 to perform 'ENTER' operation\npress 2 to 'SERVE\n");

for(i=0;i<n;i++)

{

scanf("%d",&ch);

switch(ch)

{

case 1:

{

printf("ENTER\n");

strcpy(n1," ");

scanf("%s",n1);

scanf("%f",&cp);

enter(&p,n1,cp,1);

}

break;

case 2:

{

printf("SERVE\n");

serve(&p);

}

break;

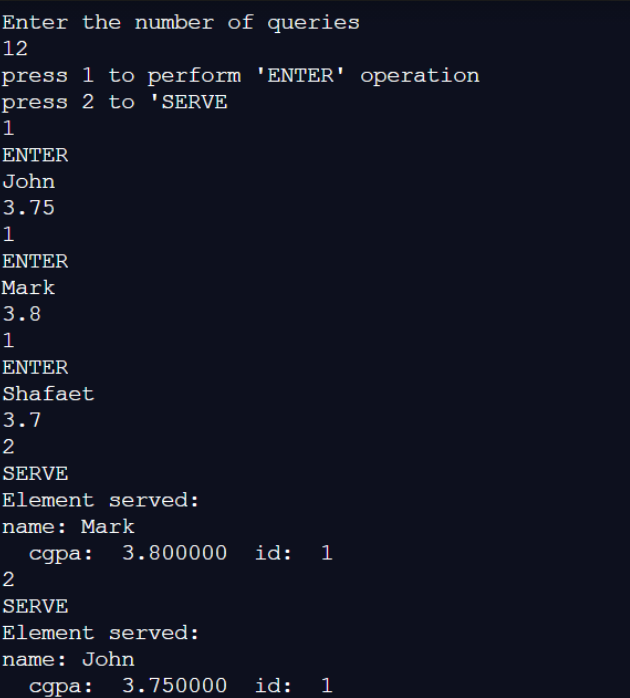
}

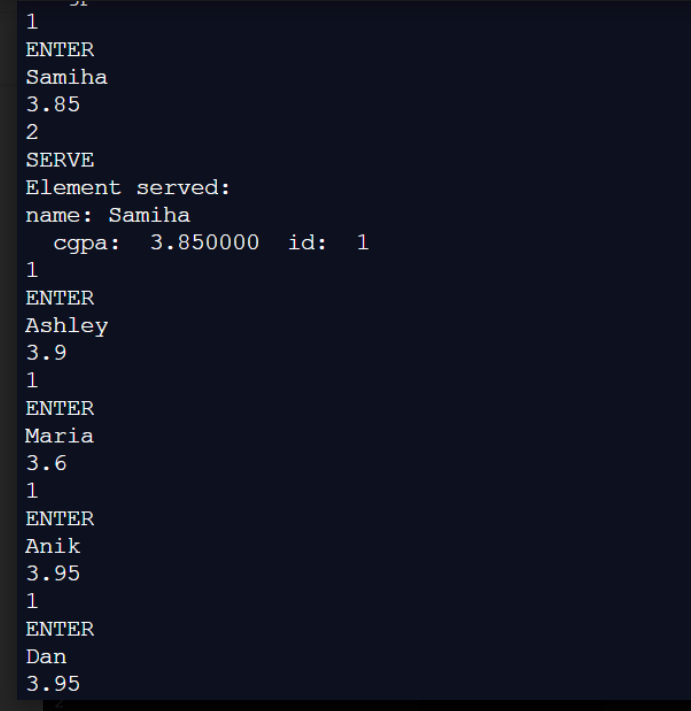
}

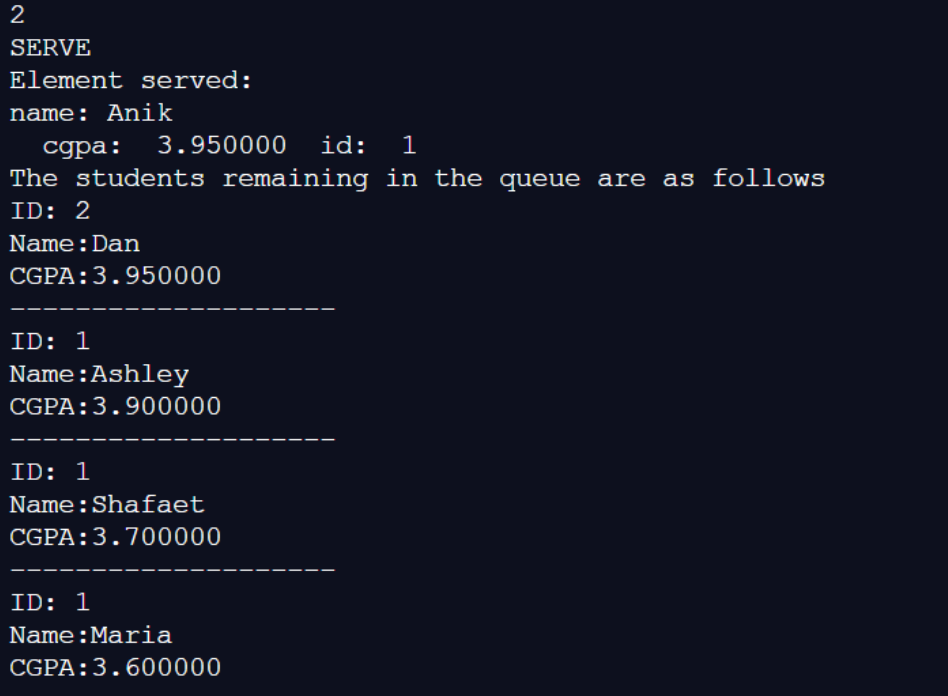
display(&p);

}

**Screenshots of Output :**

****

****

****