**L DIVISION LAB. 27 JULY 2021. 1.40-4.45 PM.**

**QUEUE PROGRAMS FOR LAB.**

1. A new Toll bay is setup at Airport Road. There are different types of vehicles passing through the toll. Vehicles are categorized as LMV and HMV. The charges for LMV passing by toll is Rs. 50 and HMV passing by toll is Rs. 100 each time vehicles pass. Apply Problem Solving Framework to perform the following:

1. Read and display the vehicles details.
2. Display only HMV vehicle details.
3. Count the LMV vehicles.

//TASK TO STUDENTS:

1. PREDICT THE TOTAL AMOUNT OF HMV VEHICLES BASED ON NUMBER OF HMV VEHICLES STANDING AT THE TOLL BAY TO PASS.

2. PREDICT THE TOTAL AMOUNT OF LMV VEHICLES BASED ON NUMBERR OF LMV VEHICLES STANDING AT THE TOLL BAY TO PASS.

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define SIZE 10

struct q

{

char vnum[SIZE][12]; //VEHICLE NUMBER

char vtype[SIZE][SIZE]; // VEHICLE TYPE. LMV OR HMV.

int cost[SIZE]; //COST BASED ON VEHICLE TYPE.

int front;

int rear;

};

typedef struct q QUEUE;

QUEUE create\_queue()

{

QUEUE q1;

q1.front = -1;

q1.rear = 0;

return q1;

}

void enqueue(QUEUE \*q1)

{

char vcnum[10], vctype[5];

if(q1->rear == SIZE)

printf("Queue Overflow\n");

else

{

printf("Enter vehicle number, vehicle type (as LMV or HMV)\n");

scanf("%s%s", vcnum, vctype);

strcpy(q1->vnum[q1->rear],vcnum);

strcpy(q1->vtype[q1->rear], vctype);

if(strcmp(q1->vtype[q1->rear],"HMV")==0) // ASSIGN COST BASED ON VEHICLE TYPE.

q1->cost[q1->rear]= 100;

else

q1->cost[q1->rear]= 50;

q1->rear ++;

}

}

void dequeue(QUEUE \*q1)

{

if(q1->rear == q1->front +1)

printf("Queue Underflow\n");

else

{

q1->front++;

printf("Vehicle moving out of toll: %s. %s. \n", q1->vnum[q1->front], q1->vtype);

}

}

void displayqueue(QUEUE \*q1) // DISPLAY ALL VEHICLES.

{

int i;

if(q1->rear == q1->front+1)

{

printf("Queue is empty\n");

}

else

{

printf("Vehicles at Toll bay are: \n");

for(i=q1->front+1; i<q1->rear; i++)

{

printf("%s %s %d\n", q1->vnum[i], q1->vtype[i], q1->cost[i]);

}

printf("\n");

}

}

void displayqueuevehicles(QUEUE \*q1) // DISPLAYS HMV AND COUNTS LMV.

{

int i, lmvcount=0;

if(q1->rear == q1->front+1)

{

printf("Queue is empty\n");

}

else

{

printf("Vehicles at Toll bay are: \n");

for(i=q1->front+1; i<q1->rear; i++)

{

if(strcmp(q1->vtype[i], "HMV")==0)

printf("%s %s %d\n", q1->vnum[i], q1->vtype[i], q1->cost[i]);

else

lmvcount++;

}

printf("\n");

}

printf("Number of LMV vehicles waiting to pass toll are: %d\n", lmvcount);

}

int main()

{

int i, n;

QUEUE q11, \*q1;

q11 = create\_queue();

q1=&q11;

printf("Enter number of vehicles waiting at Toll bay?\n");

scanf("%d",&n);

if(n>SIZE)

{

printf("Number of vehicles to enqueue are more than the size of toll bay.\n");

}

else

{

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

{

enqueue(q1);

}

displayqueue(q1);

printf("HMV vehicles: \n");

displayqueuevehicles(q1);

}

return 0;

}

//TASK TO STUDENTS:

1. PREDICT THE TOTAL AMOUNT OF HMV VEHICLES BASED ON NUMBER OF HMV VEHICLES STANDING AT THE TOLL BAY TO PASS.

2. PREDICT THE TOTAL AMOUNT OF LMV VEHICLES BASED ON NUMBERR OF LMV VEHICLES STANDING AT THE TOLL BAY TO PASS.

**//1. First function. Function call to be added in main.**

void predicthmvcost(QUEUE \*q1)

{

int i, totalhmvcost=0;

if(q1->rear == q1->front+1)

{

printf("Queue is empty\n");

}

else

{

for(i=q1->front+1; i<q1->rear; i++)

{

if(strcmp(q1->vtype[i], "HMV")==0)

totalhmvcost = totalhmvcost + q1->cost[i];

}

printf("Predicted Total cost of HMV is %d\n", totalhmvcost);

}

}

**//2. Second function. Function call to be added in main.**

void predictlmvcost(QUEUE \*q1)

{

int i, totallmvcost=0;

if(q1->rear == q1->front+1)

{

printf("Queue is empty\n");

}

else

{

for(i=q1->front+1; i<q1->rear; i++)

{

if(strcmp(q1->vtype[i], "LMV")==0)

totallmvcost = totallmvcost + q1->cost[i];

}

printf("Predicted Total cost of LMV is %d\n", totallmvcost);

}

}

2. Tirumala [temple](http://en.wikipedia.org/wiki/Tirumala_Venkateswara_Temple)is one of the most visited places of worship in the world. As the number of pilgrims who visit the temple each day is very high, the head of the temple should keep monitoring the pilgrims who are visiting the temple and pilgrims are allowed with preference to worship. Today is another crowdie day, Head of the temple started to apply the rule to manage the crowd, i.e, if the age of arriving pilgrim is greater than the age of pilgrim who is ready to worship, the arriving pilgrim will be allowed to join in the middle of the line (count of pilgrims before the pilgrim whose age is greater than the first pilgrim divided by 2)+1 otherwise will be allowed to join at the end of the line. Please help the head of the temple to manage the pilgrims.

Numbers represent the age of the pilgrims.

**Ex 1:**

50 25 34 54 12 22 60

50 54 25 60 34 12 22

Suppose the age of pilgrim to worship is 50, the next arriving pilgrim’s age is 25. Therefore he joins at the end of the line (list is 50 25).

The next arriving pilgrim’s age is 34, he also joins at the end of the line (list is 50 25 34).

The next arriving pilgrim’s age is 54, which is greater than the age of pilgrim to worship, therefore he joins at the middle of the line - (3 arrived pilgrims /2) +1 (list becomes 50 54 25 34).

The process continues until all the pilgrims arrive.

**Ex 2:**

46 22 34 53 80 44

46 53 80 22 34 44

**Constraints:**

2<=**N**<=100

10<=**Age**<=100

**Input format:**

First-line indicates the size and the second line indicates the age of pilgrims.

**Output format:**

First-line indicates the pilgrims after applying the rule.

|  |  |
| --- | --- |
| **Sample Input 0:**  7  50 25 34 54 12 22 60  **Sample Output 0:**  50 54 25 60 34 12 22  **Sample Input 1:**  6  46 22 34 53 80 44  **Sample Output 1:**  46 53 80 22 34 44 | **Sample Input 2:**  7  60 40 56 69 55 23 62  **Sample Output 2:**  60 69 40 62 56 55 23 |

#include <stdlib.h>

#include<stdio.h>

typedef struct node

{

int data;

struct node \*next;

}\*List;

int count=0;

List create();

List insertrear(List,int);

void display(List);

List insertatpos(List first,int item,int pos);

int main()

{

List L=NULL;

int ele,ch,m,i;

scanf("%d",&ch);

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

{

scanf("%d",&ele);

if(count==0)

L=insertrear(L,ele);

else

{

if(L->data<ele)

{

m=count/2;

L=insertatpos(L,ele,m);

}

else

L=insertrear(L,ele);

}

}

display(L);

return 0;

}

List create()

{

List x;

x=malloc(sizeof(struct node));

count++;

x->next=NULL;

return x;

}

List insertrear(List L,int x)

{

List temp,p=L;

temp=create();

temp->data=x;

if(L==NULL)

{

L=temp;

}

else{

while(p->next!=NULL)

{

p=p->next;

}

p->next=temp;

}

return L;

}

void display(List L)

{

if(L==NULL)

{

printf("List is empty\n");

}

else

{

while(L!=NULL)

{

printf("%d ",L->data);

L=L->next;

}

}

}

List insertatpos(List first,int item,int pos)

{

List t,prev,cur;

int c=0;

t=create( );

t->data=item;

cur=first;

prev=NULL;

if(pos==0)

{

first->next=t;

//return first;

}

else{

while(cur->next!=NULL && c!=pos)

{

c++;

prev=cur;

cur=cur->next;

}

prev->next=t;

t->next=cur;

}

return first;

}