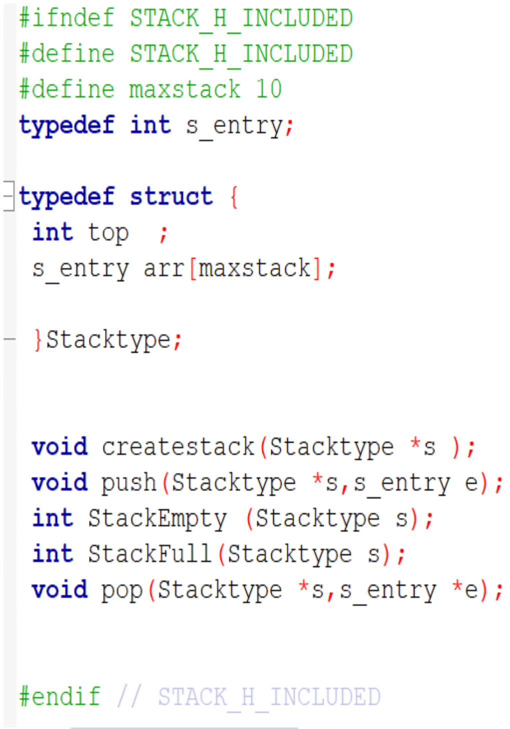
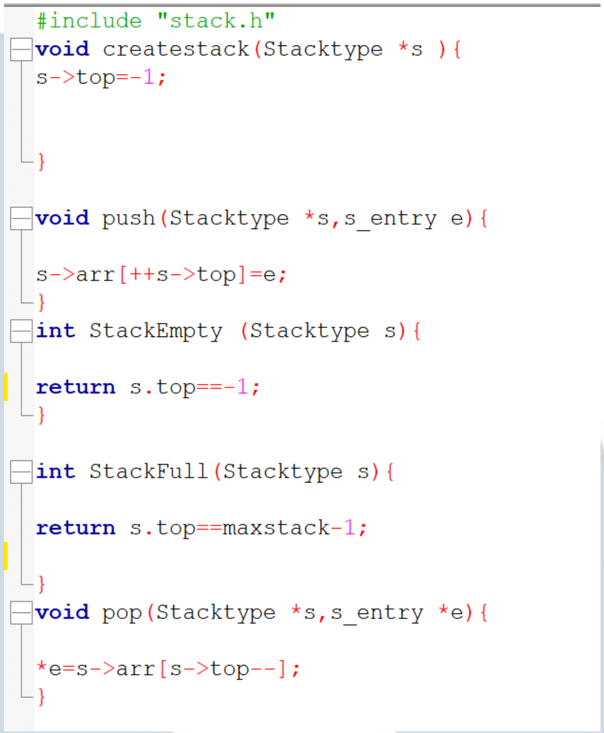
Stack sheet2 solution

STACK.H



```
#ifndef STACK_H_INCLUDED
#define STACK_H_INCLUDED
#define maxstack 10
typedef int s_entry;

typedef struct {
 int top   ;
 s_entry arr[maxstack];

 }Stacktype;


 void createstack(Stacktype *s );
 void push(Stacktype *s,s_entry e);
 int StackEmpty (Stacktype s);
 int StackFull(Stacktype s);
 void pop(Stacktype *s,s_entry *e);


#endif // STACK_H_INCLUDED
```

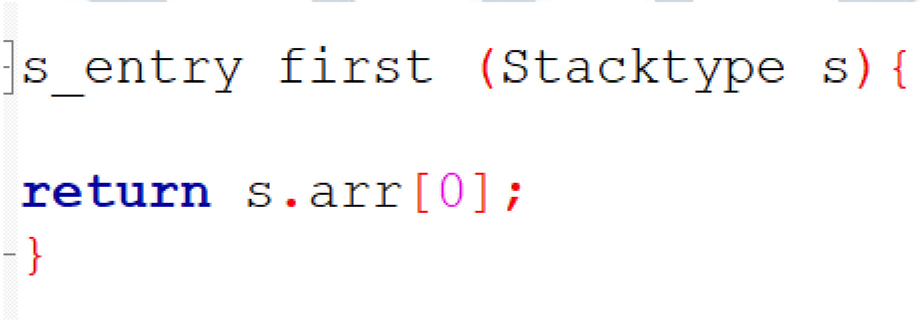
STACK.C



```
#include "stack.h"
void createstack(Stacktype *s ){
s->top=-1;


}

void push(Stacktype *s,s_entry e){

s->arr[++s->top]=e;
}
int StackEmpty (Stacktype s){

return s.top==-1;
}

int StackFull(Stacktype s){

return s.top==maxstack-1;

}
void pop(Stacktype *s,s_entry *e){

*e=s->arr[s->top--];
}
```

1. **Write a function that returns the first element entered to a stack. (implementation level) =stack.c**

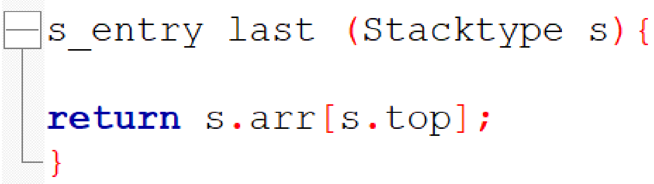
SOLUTION



```
s_entry first (Stacktype s){

return s.arr[0];
}
```

2. Write a function that returns a copy from the last element in a stack. (implementation level) =stack.c

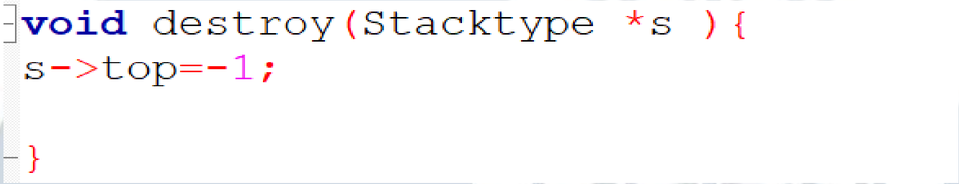
SOLUTION



```
s_entry last (Stacktype s){

return s.arr[s.top];
}
```

3.Write a function to destroy a stack. (implementation level) =stack.c

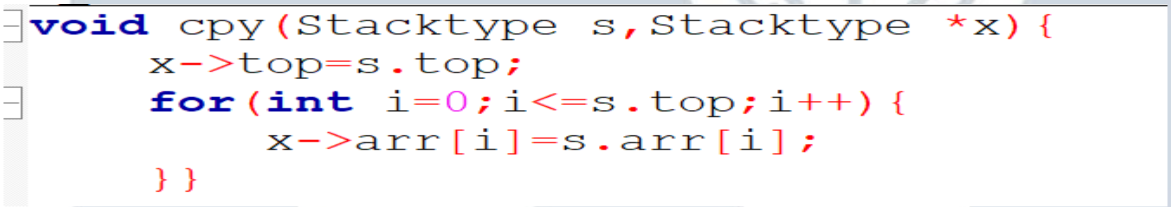
SOLUTION



```
void destroy(Stacktype *s ){
s->top=-1;

}
```

4. Write a function to copy a stack to another. (implementation level) =stack.c

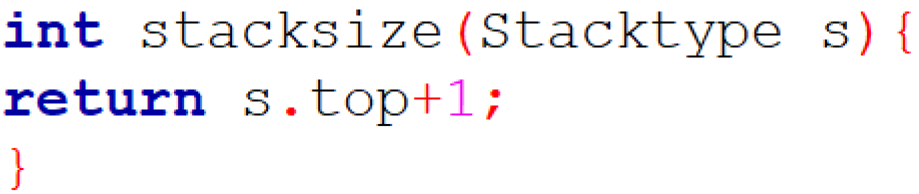
SOLUTION



```
void cpy(Stacktype s,Stacktype *x){
    x->top=s.top;
    for(int i=0;i<=s.top;i++){
        x->arr[i]=s.arr[i];
    }}
```

5. Write a function to return the size of a stack (implementation level) =stack.c

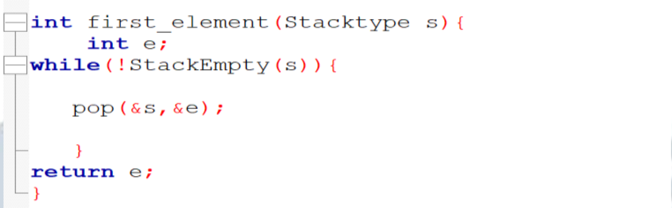
SOLUTION



```
int stacksize(Stacktype s){
return s.top+1;
}
```

6. Write a function that returns the first element entered to a stack. (userlevel)=main.c

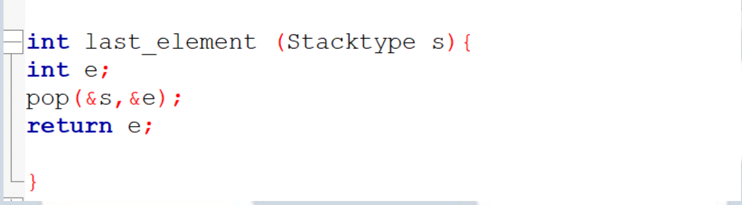
SOLUTION



```
int first_element(Stacktype s){
    int e;
while(!StackEmpty(s)){

    pop(&s,&e);

    }
return e;
}
```

7. Write a function that returns a copy from the last element in a stack. (user level) =main.c

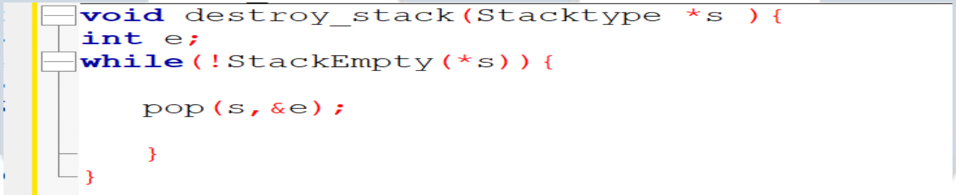
SOLUTION



```
int last_element (Stacktype s){
int e;
pop(&s,&e);
return e;

}
```

8. Write a function to destroy a stack. (user level) =main.c

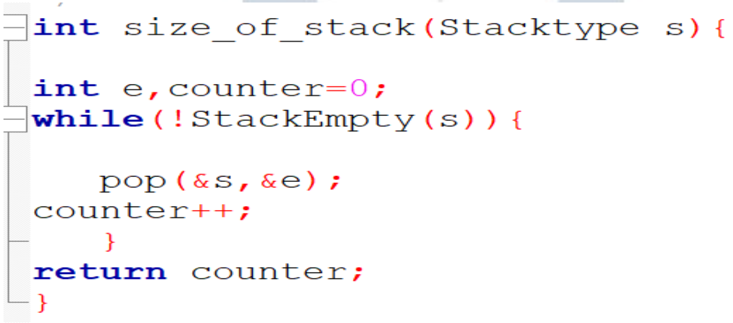
SOLUTION



```
void destroy_stack(Stacktype *s ){
int e;
while(!StackEmpty(*s)){

    pop(s,&e);

    }
}
```

9. Write a function to return the size of a stack (user level) =main.c

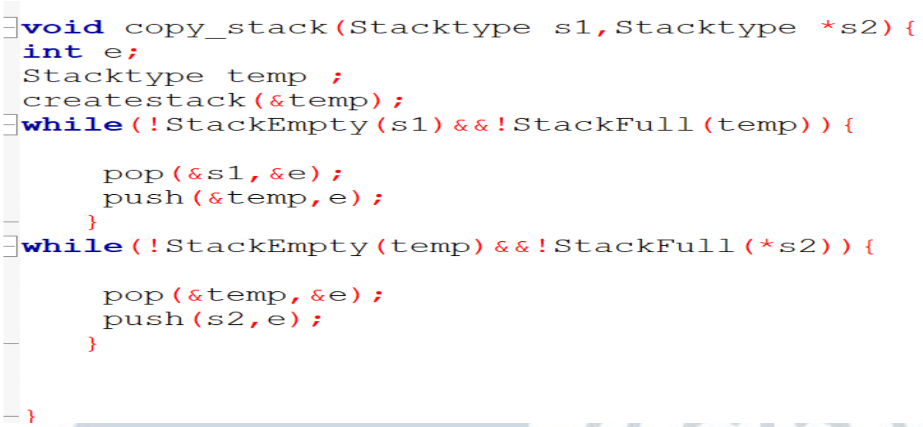
SOLUTION



```
int size_of_stack(Stacktype s){

int e,counter=0;
while(!StackEmpty(s)){

    pop(&s,&e);
counter++;
    }
return counter;
}
```

10. Write a function to copy a stack to another. (user level) =main.c

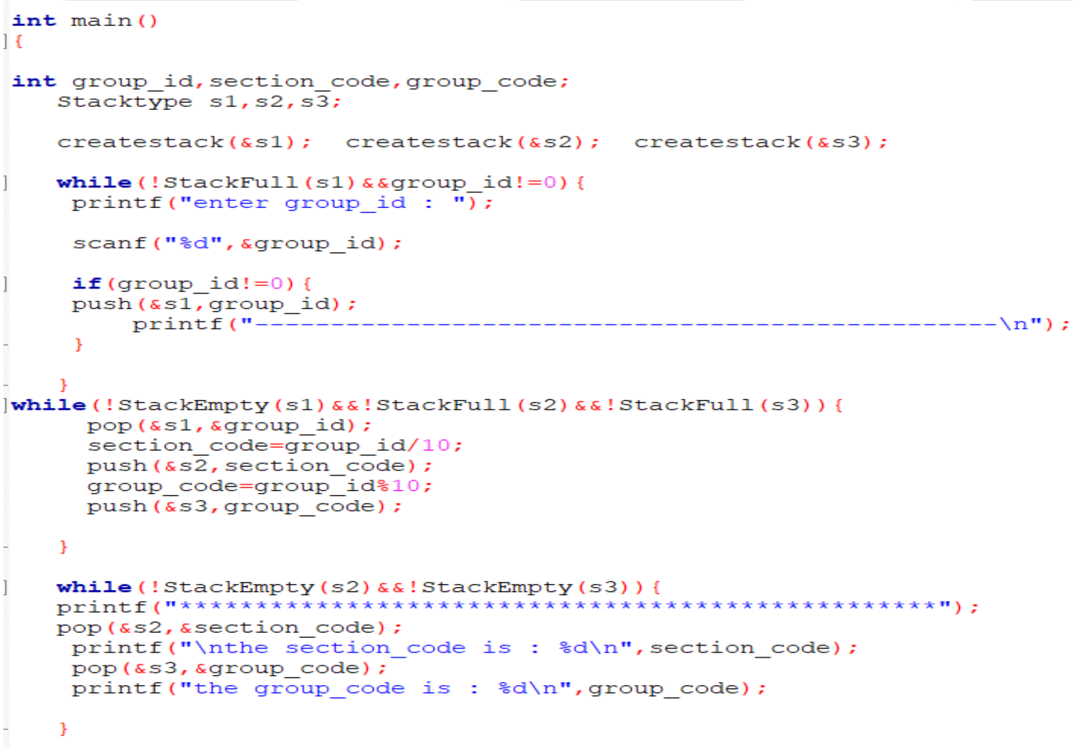
SOLUTION



```
void copy_stack(Stacktype s1,Stacktype *s2){
int e;
Stacktype temp ;
createstack(&temp);
while(!StackEmpty(s1)&&!StackFull(temp)){

     pop(&s1,&e);
     push(&temp,e);
    }
while(!StackEmpty(temp)&&!StackFull(*s2)){

     pop(&temp,&e);
     push(s2,e);
    }


}
```

11. We (as a user for StackADT) have a stack holding group_ids. Each group_id consists of two parts section code and group code within his section. Number of groups inside the section is maximum 10. (section_code=group_id/10, group_code=group_id%10). Construct two stacks; one stack holds section codes while the other holds group codes.

SOLUTION

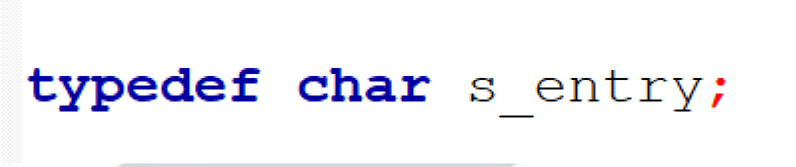


```
int main()
{

int group_id,section_code,group_code;
   Stacktype s1,s2,s3;

   createstack(&s1);  createstack(&s2);  createstack(&s3);

   while(!StackFull(s1)&&group_id!=0){
    printf("enter group_id :  ");

    scanf("%d",&group_id);

    if(group_id!=0){
    push(&s1,group_id);
        printf("-------------------------------------------------\n");
    }

   }
while(!StackEmpty(s1)&&!StackFull(s2)&&!StackFull(s3)){
     pop(&s1,&group_id);
     section_code=group_id/10;
     push(&s2,section_code);
     group_code=group_id%10;
     push(&s3,group_code);

   }

   while(!StackEmpty(s2)&&!StackEmpty(s3)){
   printf("***************************************************");
   pop(&s2,&section_code);
    printf("\nthe section_code is : %d\n",section_code);
    pop(&s3,&group_code);
    printf("the group_code is : %d\n",group_code);

   }
```

12. Use a stack structure to check the balance and ordering between various parentheses.

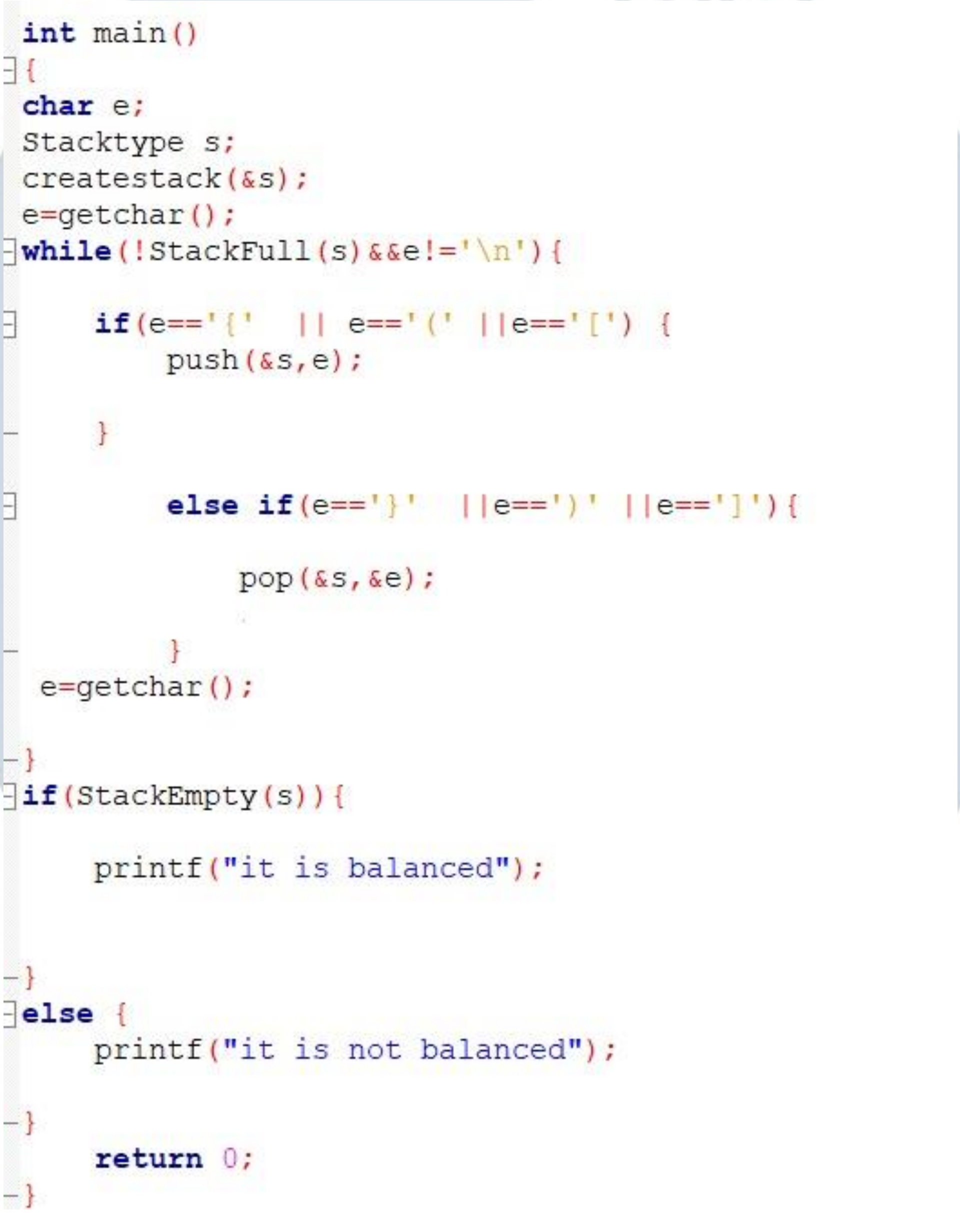
SOLUTION

Stack.h (changes)



```
typedef char s_entry;
```

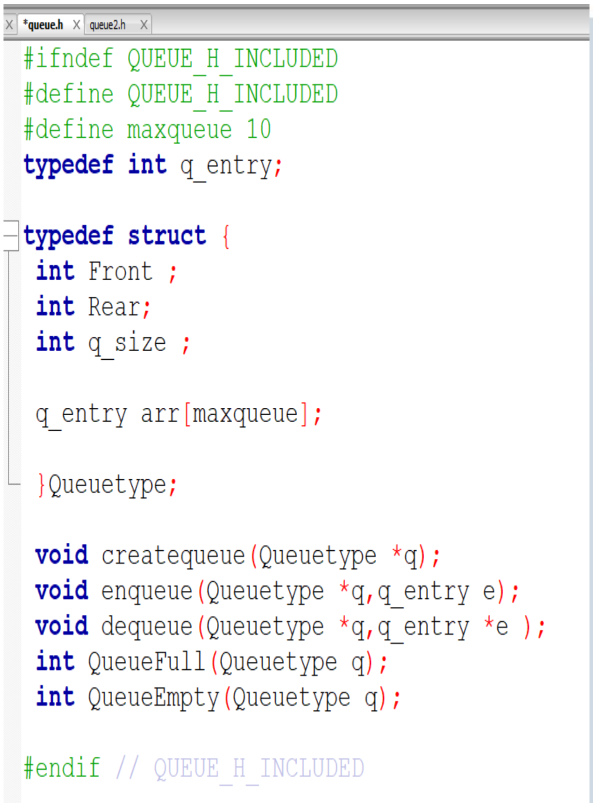
Main.c



```
int main()
{
char e;
Stacktype s;
createstack(&s);
e=getchar();
while(!StackFull(s)&&e!='\n'){

    if(e=='{'  || e=='(' ||e=='[') {
        push(&s,e);

    }

        else if(e=='}'  ||e==')' ||e==']'){

            pop(&s,&e);

        }
 e=getchar();

}
if(StackEmpty(s)){

    printf("it is balanced");


}
else {
    printf("it is not balanced");

}
    return 0;
}
```

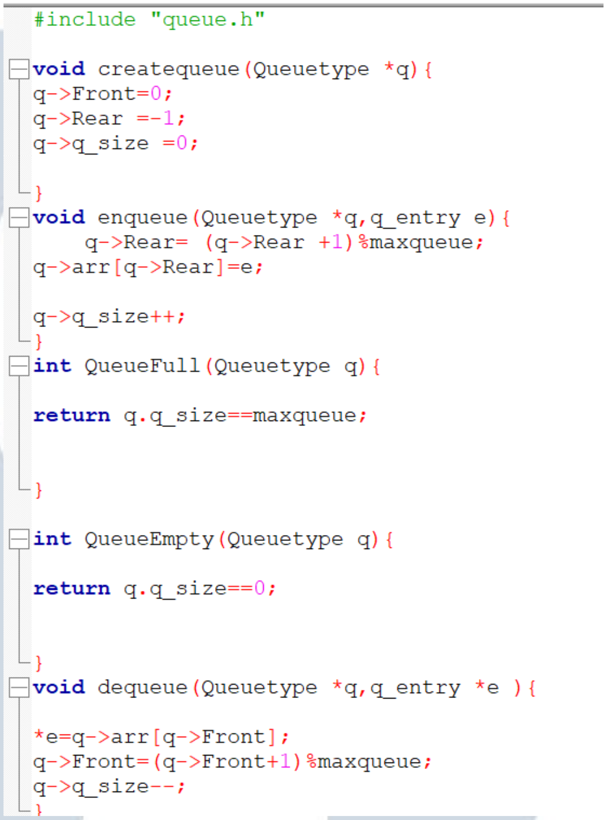
QUEUE SHEET 3 SOLUTIONS

QUEUE .h



```
#ifndef QUEUE_H_INCLUDED
#define QUEUE_H_INCLUDED
#define maxqueue 10
typedef int q_entry;

typedef struct {
 int Front ;
 int Rear;
 int q_size ;

 q_entry arr[maxqueue];

 }Queuetype;

 void createqueue(Queuetype *q);
 void enqueue(Queuetype *q,q_entry e);
 void dequeue(Queuetype *q,q_entry *e );
 int QueueFull(Queuetype q);
 int QueueEmpty(Queuetype q);

#endif // QUEUE_H_INCLUDED
```

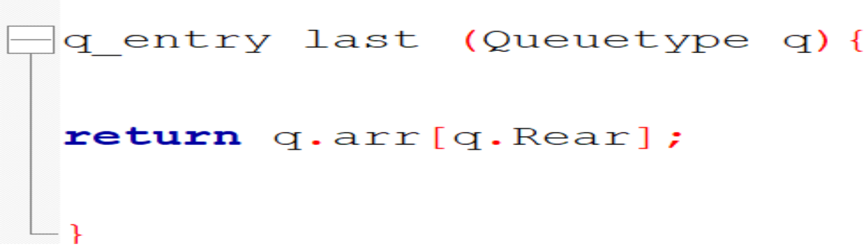
QUEUE.C



```
#include "queue.h"

void createqueue(Queuetype *q){
q->Front=0;
q->Rear =-1;
q->q_size =0;

}
void enqueue(Queuetype *q,q_entry e){
    q->Rear= (q->Rear +1)%maxqueue;
q->arr[q->Rear]=e;

q->q_size++;
}
int QueueFull(Queuetype q){

return q.q_size==maxqueue;


}

int QueueEmpty(Queuetype q){

return q.q_size==0;


}
void dequeue(Queuetype *q,q_entry *e ){

*e=q->arr[q->Front];
q->Front=(q->Front+1)%maxqueue;
q->q_size--;
}
```

1. Write a function that returns the last element in a queue. (implementation level) = QUEUE.C

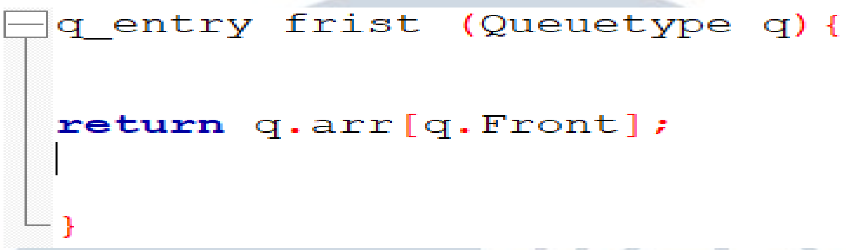
SOLUTION



```
q_entry last (Queuetype q) {

return q.arr[q.Rear];


}
```

2. Write a function that returns a copy from the first element in a queue. (implementation level) = QUEUE.C

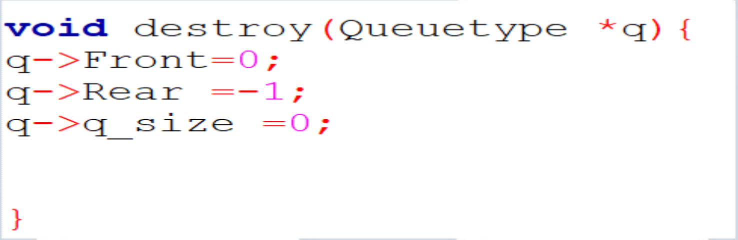
SOLUTION



```
q_entry frist (Queuetype q){

return q.arr[q.Front];

}
```

3. Write a function to destroy a queue (implementation level) = QUEUE.C

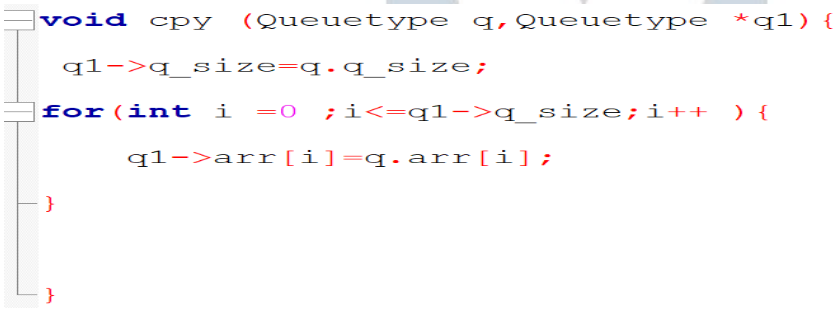
SOLUTION



```
void destroy(Queuetype *q){
q->Front=0;
q->Rear =-1;
q->q_size =0;


}
```

4. Write a function to copy a queue to another. (implementation level) = QUEUE.C

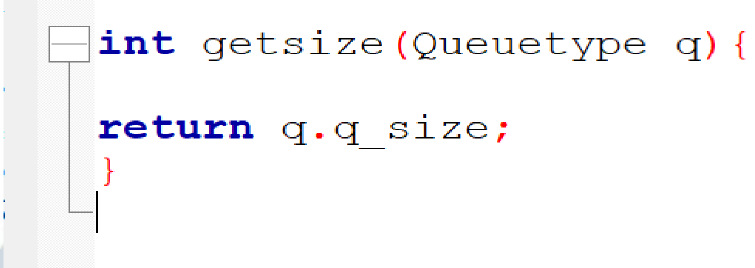
SOLUTION



```
void cpy (Queuetype q,Queuetype *q1){
 q1->q_size=q.q_size;
for(int i =0 ;i<=q1->q_size;i++ ){
     q1->arr[i]=q.arr[i];
}


}
```

5. Write a function to return the size of a queue (implementation level) = QUEUE.C

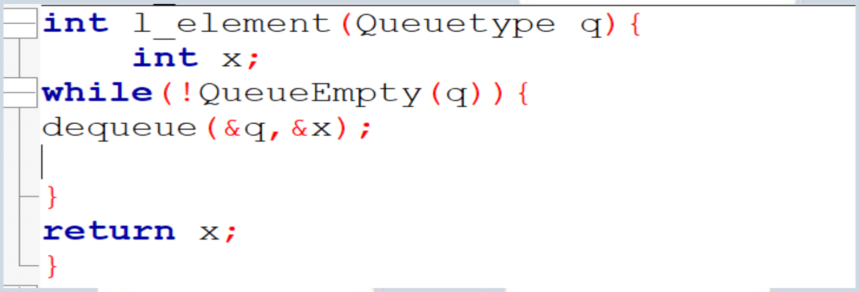
SOLUTION



```
int getsize(Queuetype q){

return q.q_size;
}
```

6. Write a function that returns the last element in a queue. (user level)=MAIN.C

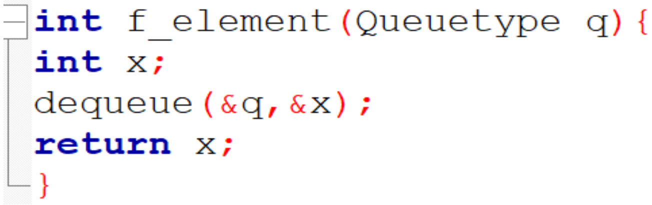
SOLUTION



```
int l_element(Queuetype q){
    int x;
while(!QueueEmpty(q)){
dequeue(&q,&x);

}
return x;
}
```

7. Write a function that returns a copy from the first element in a queue. (user level))=MAIN.C

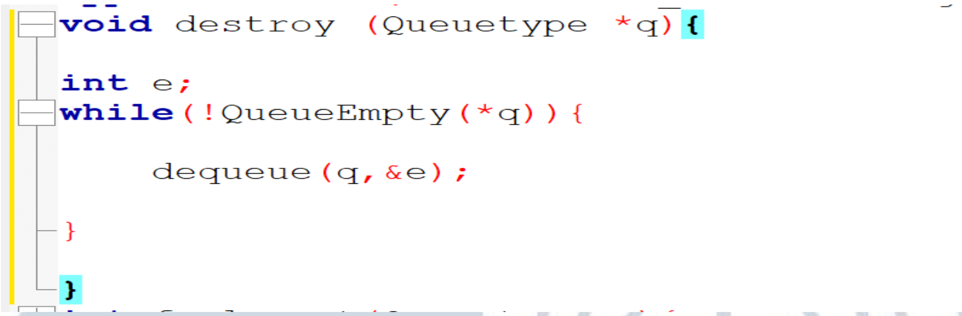
SOLUTION



```
int f_element(Queuetype q){
int x;
dequeue(&q,&x);
return x;
}
```

8. Write a function to destroy a queue (user level) = MAIN.C

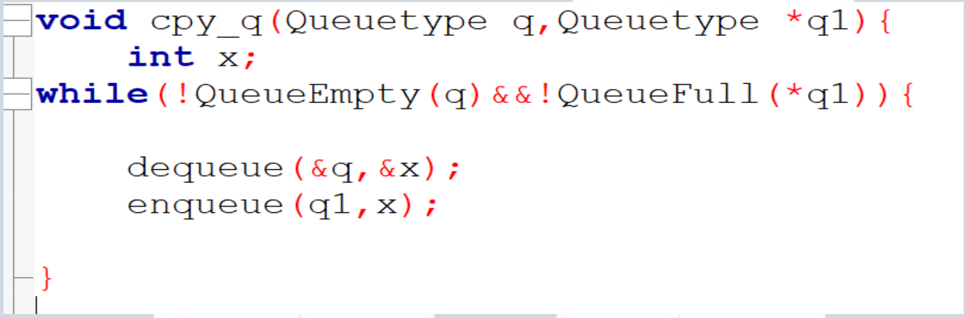
SOLUTION



```
void destroy  (Queuetype  *q){

int e;
while(!QueueEmpty(*q)){

     dequeue(q,&e);

}

}
```

9. Write a function to copy a queue to another. (user level) = MAIN.C

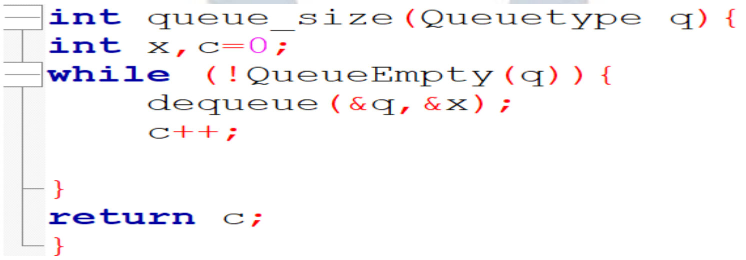
SOLUTION



```
void cpy_q(Queuetype q,Queuetype *q1){
    int x;
while(!QueueEmpty(q)&&!QueueFull(*q1)){

    dequeue(&q,&x);
    enqueue(q1,x);

}
```

10. Write a function to return the size of a queue (user level) = MAIN.C

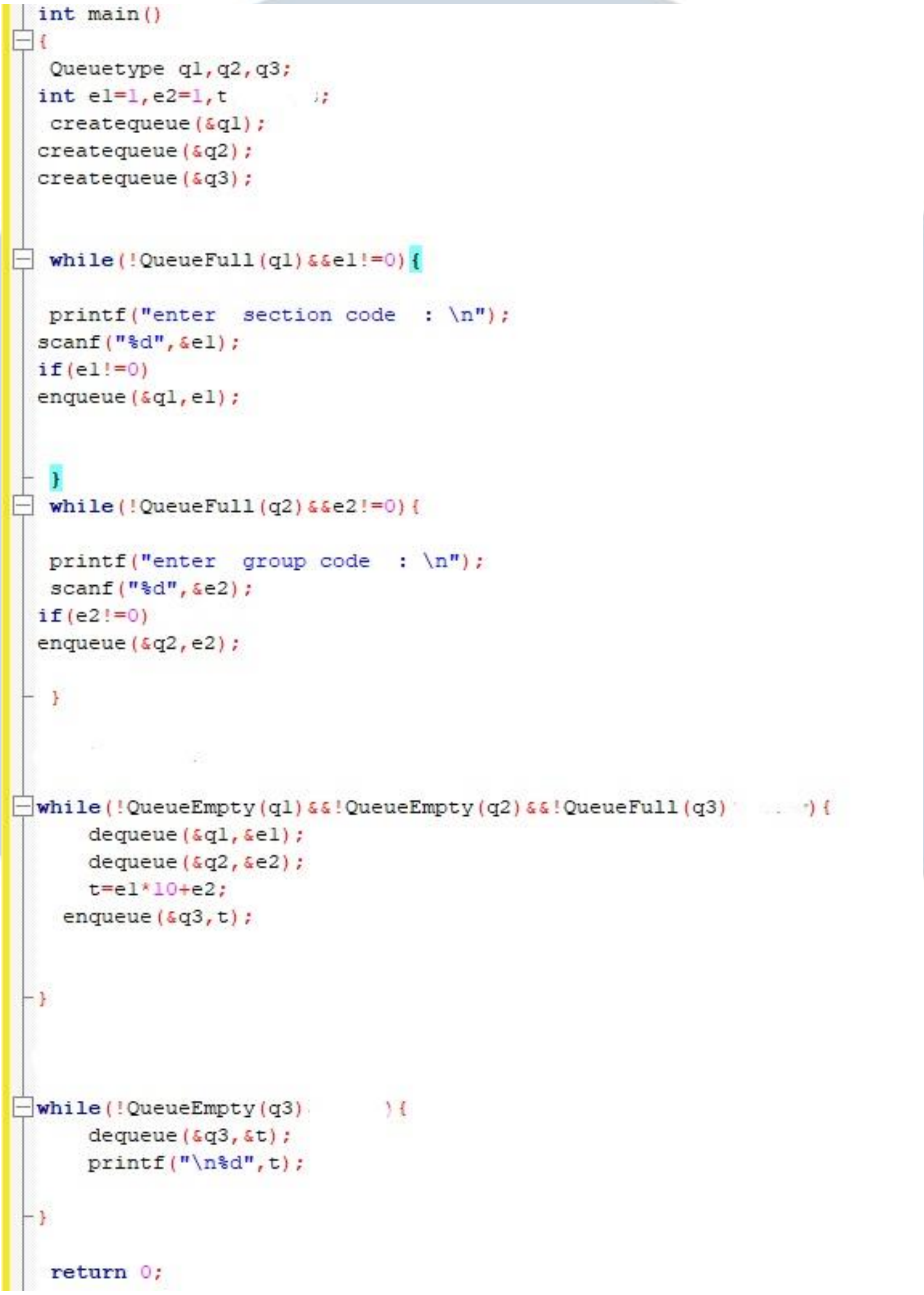
SOLUTION



```
int queue_size(Queuetype q){
int x,c=0;
while  (!QueueEmpty(q)){
    dequeue(&q,&x);
    c++;

}
return c;
}
```

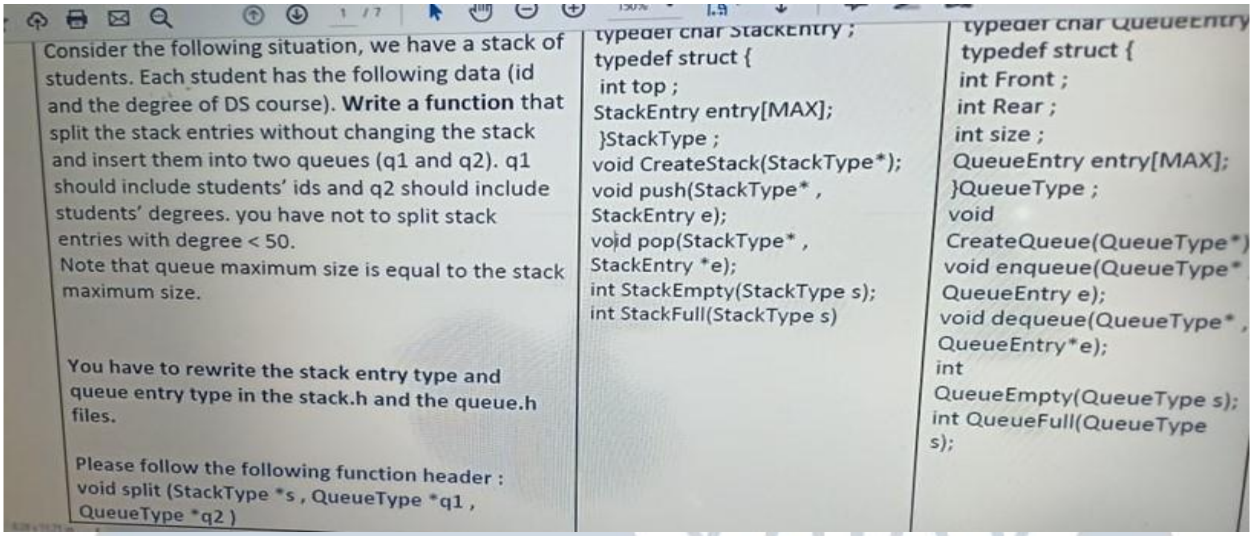
11. We (as a user for QueueADT) have two filled queues; the first queue holds section code while the other holds group code (where number of groups inside the section is maximum 10). Merge those numbers (section code*10+group code) in a newly created queue.

SOLUTION



```
int main()
{
 Queuetype q1,q2,q3;
int e1=1,e2=1,t          ;
 createqueue(&q1);
createqueue(&q2);
createqueue(&q3);


 while(!QueueFull(q1)&&e1!=0){

 printf("enter  section code  : \n");
scanf("%d",&e1);
if(e1!=0)
enqueue(&q1,e1);


 }
 while(!QueueFull(q2)&&e2!=0){

 printf("enter  group code  : \n");
 scanf("%d",&e2);
if(e2!=0)
enqueue(&q2,e2);

 }



while(!QueueEmpty(q1)&&!QueueEmpty(q2)&&!QueueFull(q3)      ){
    dequeue(&q1,&e1);
    dequeue(&q2,&e2);
    t=e1*10+e2;
  enqueue(&q3,t);


}



while(!QueueEmpty(q3)       ){
    dequeue(&q3,&t);
    printf("\n%d",t);

}

 return 0;
```

Quizes 2 solutions



Consider the following situation, we have a stack of students. Each student has the following data (id and the degree of DS course). **Write a function** that split the stack entries without changing the stack and insert them into two queues (q1 and q2). q1 should include students' ids and q2 should include students' degrees. you have not to split stack entries with degree < 50.

Note that queue maximum size is equal to the stack maximum size.

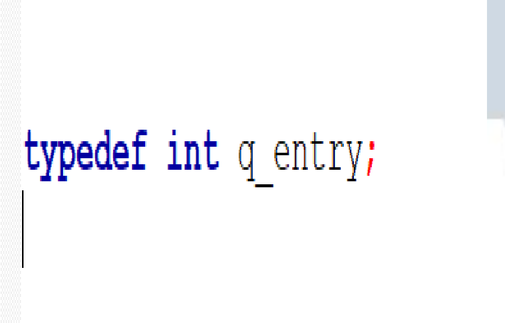
You have to rewrite the stack entry type and queue entry type in the stack.h and the queue.h files.

Please follow the following function header :
void split (StackType *s , QueueType *q1 ,
QueueType *q2)

```
typedef char StackEntry ;
typedef struct {
 int top ;
StackEntry entry[MAX];
 }StackType ;
void CreateStack(StackType*);
void push(StackType* ,
StackEntry e);
void pop(StackType* ,
StackEntry *e);
int StackEmpty(StackType s);
int StackFull(StackType s)
```

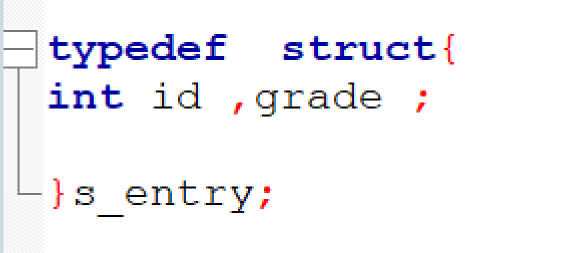
```
typedef char QueueEntry
typedef struct {
int Front ;
int Rear ;
int size ;
QueueEntry entry[MAX];
}QueueType ;
void
CreateQueue(QueueType*)
void enqueue(QueueType*
QueueEntry e);
void dequeue(QueueType* ,
QueueEntry*e);
int
QueueEmpty(QueueType s);
int QueueFull(QueueType
s);
```

QUEUE .h (changes)



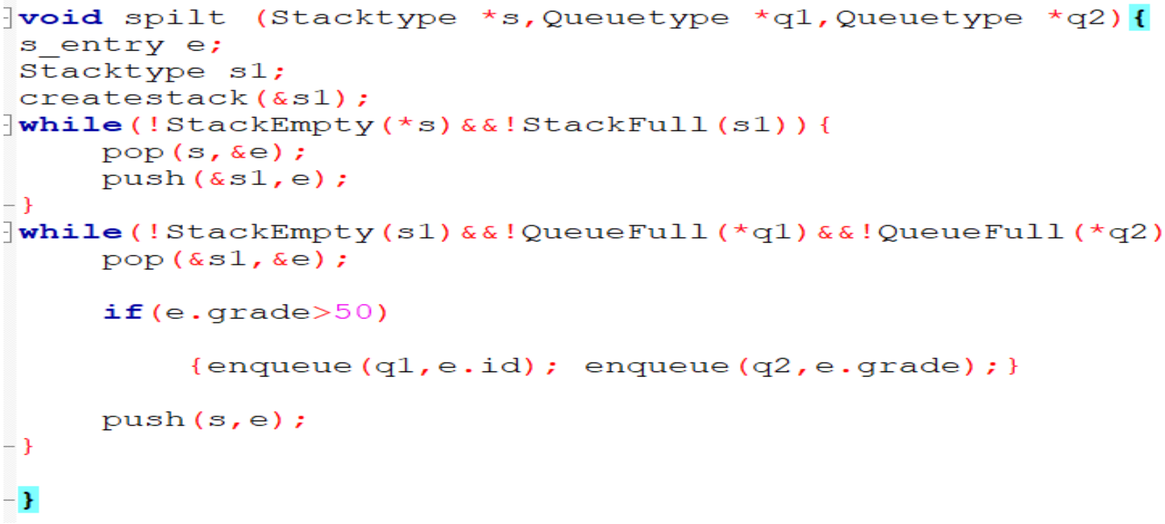
```
typedef int q_entry;
```

STACK .h (changes)

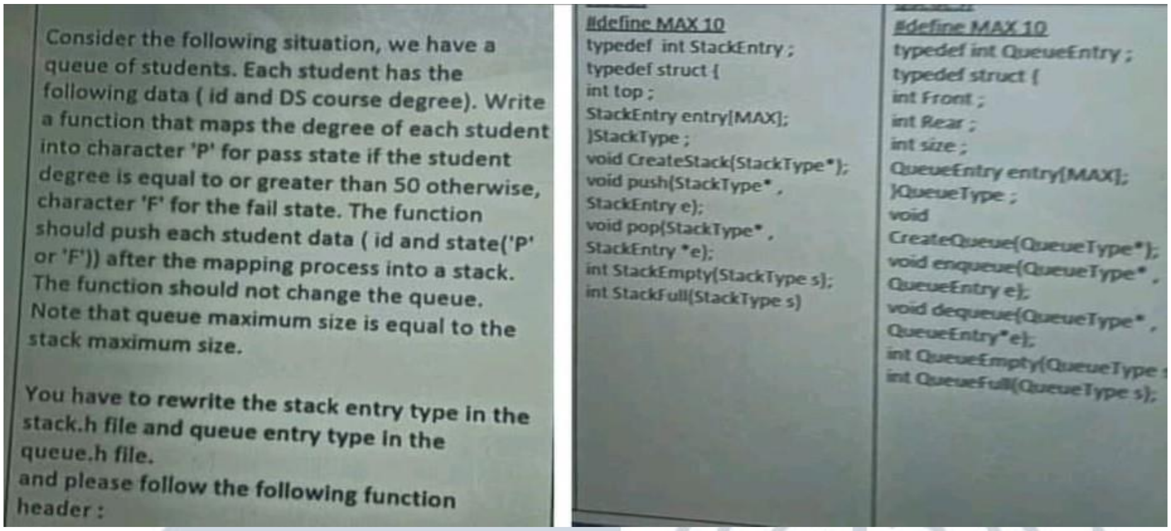


```
typedef  struct{
int id ,grade ;

}s_entry;
```

FUNCTION (USER LEVEL)=MAIN.C



```
void spilt  (Stacktype *s,Queuetype *q1,Queuetype *q2){
s_entry e;
Stacktype s1;
createstack(&s1);
while(!StackEmpty(*s)&&!StackFull(s1)){
      pop(s,&e);
      push(&s1,e);
}
while(!StackEmpty(s1)&&!QueueFull(*q1)&&!QueueFull(*q2)){
      pop(&s1,&e);

      if(e.grade>50)

            {enqueue(q1,e.id); enqueue(q2,e.grade);}

      push(s,e);
}

}
```



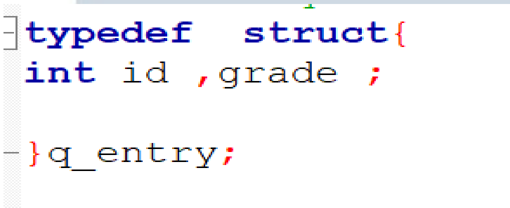
Consider the following situation, we have a queue of students. Each student has the following data (id and DS course degree). Write a function that maps the degree of each student into character 'P' for pass state if the student degree is equal to or greater than 50 otherwise, character 'F' for the fail state. The function should push each student data (id and state('P' or 'F')) after the mapping process into a stack. The function should not change the queue. Note that queue maximum size is equal to the stack maximum size.

You have to rewrite the stack entry type in the stack.h file and queue entry type in the queue.h file.
and please follow the following function header :

```
#define MAX 10
typedef  int StackEntry ;
typedef struct {
int top ;
StackEntry entry[MAX];
}StackType ;
void CreateStack(StackType*);
void push(StackType* ,
StackEntry e);
void pop(StackType* ,
StackEntry *e);
int StackEmpty(StackType s);
int StackFull(StackType s)
```

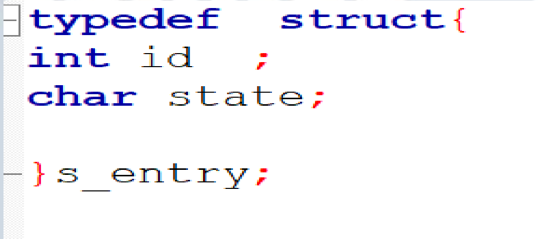
```
#define MAX 10
typedef int QueueEntry ;
typedef struct {
int Front ;
int Rear ;
int size ;
QueueEntry entry[MAX];
}QueueType ;
void
CreateQueue(QueueType*);
void enqueue(QueueType* ,
QueueEntry e);
void dequeue(QueueType* ,
QueueEntry*e);
int QueueEmpty(QueueType
int QueueFull(QueueType s);
```

QUEUE .h (changes)



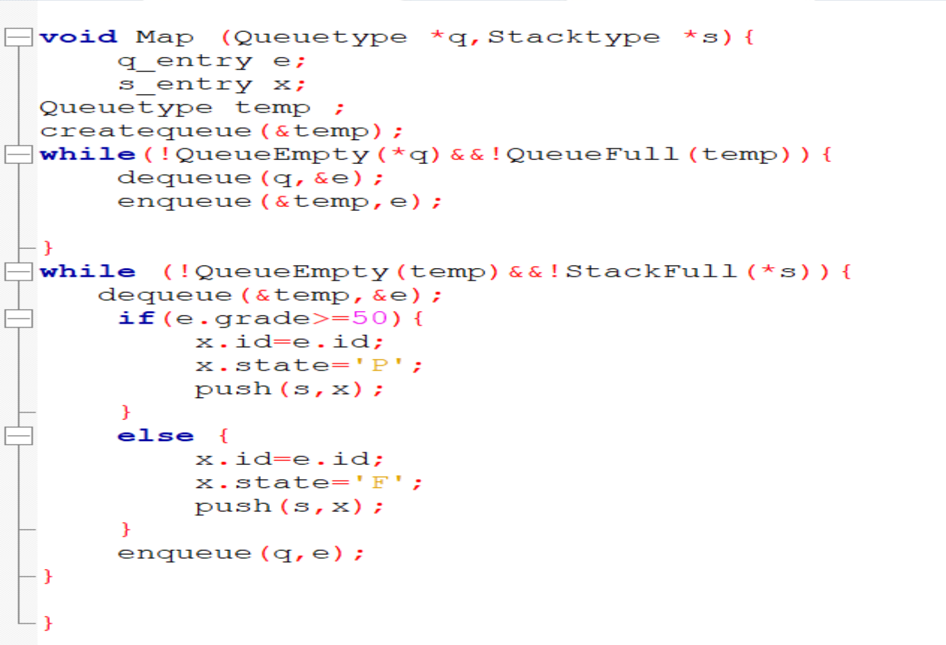
```
typedef  struct{
int id ,grade ;

}q_entry;
```

STACK .h (changes)

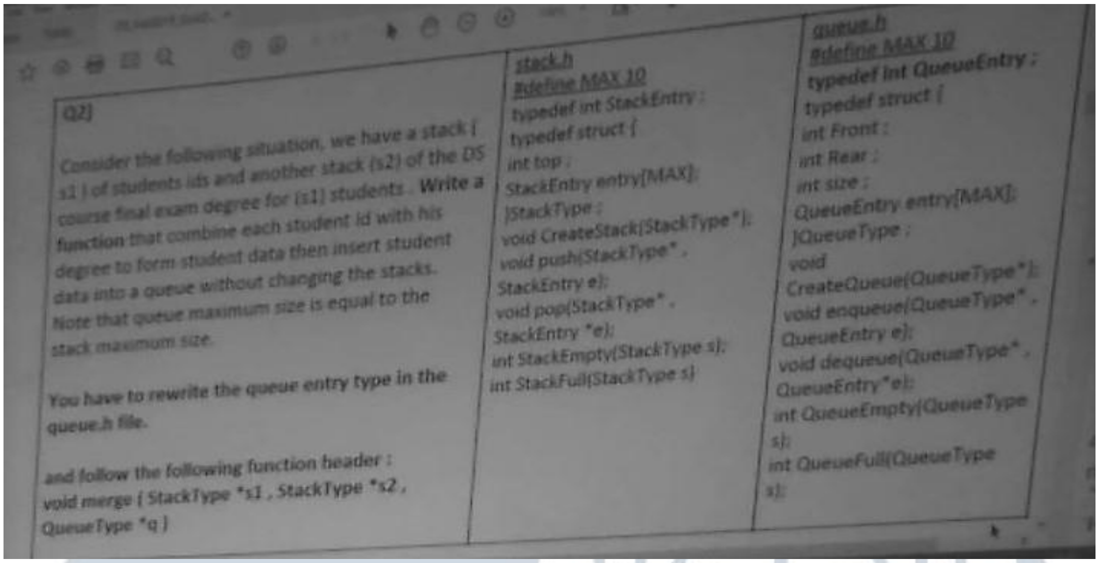


```
typedef  struct{
int id  ;
char state;

}s_entry;
```

FUNCTION (USER LEVEL)=MAIN.C



```
void Map  (Queuetype *q,Stacktype *s){
    q_entry e;
    s_entry x;
Queuetype temp ;
createqueue(&temp);
while(!QueueEmpty(*q)&&!QueueFull(temp)){
    dequeue(q,&e);
    enqueue(&temp,e);

}
while  (!QueueEmpty(temp)&&!StackFull(*s)){
   dequeue(&temp,&e);
    if(e.grade>=50){
        x.id=e.id;
        x.state='P';
        push(s,x);
    }
    else  {
        x.id=e.id;
        x.state='F';
        push(s,x);
    }
    enqueue(q,e);
}

}
```



Q2)

Consider the following situation, we have a stack (s1) of students ids and another stack (s2) of the DS course final exam degree for (s1) students . **Write a function** that combine each student id with his degree to form student data then insert student data into a queue without changing the stacks. Note that queue maximum size is equal to the stack maximum size.

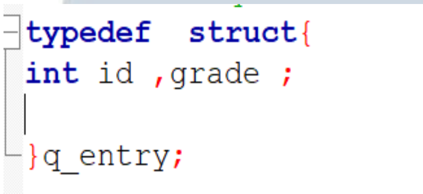
You have to rewrite the queue entry type in the queue.h file.

and follow the following function header :
void merge (StackType *s1 , StackType *s2 , QueueType *q)

```
stack.h
#define MAX 10
typedef int StackEntry ;
typedef struct {
int top ;
StackEntry entry[MAX];
}StackType ;
void CreateStack(StackType*);
void push(StackType* ,
StackEntry e);
void pop(StackType* ,
StackEntry *e);
int StackEmpty(StackType s);
int StackFull(StackType s)
```

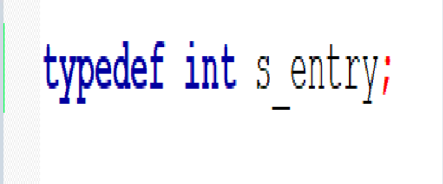
```
queue.h
#define MAX 10
typedef int QueueEntry ;
typedef struct {
int Front ;
int Rear ;
int size ;
QueueEntry entry[MAX];
}QueueType ;
void
CreateQueue(QueueType*);
void enqueue(QueueType* ,
QueueEntry e);
void dequeue(QueueType* ,
QueueEntry*e);
int QueueEmpty(QueueType
s);
int QueueFull(QueueType
s);
```

QUEUE .h (changes)



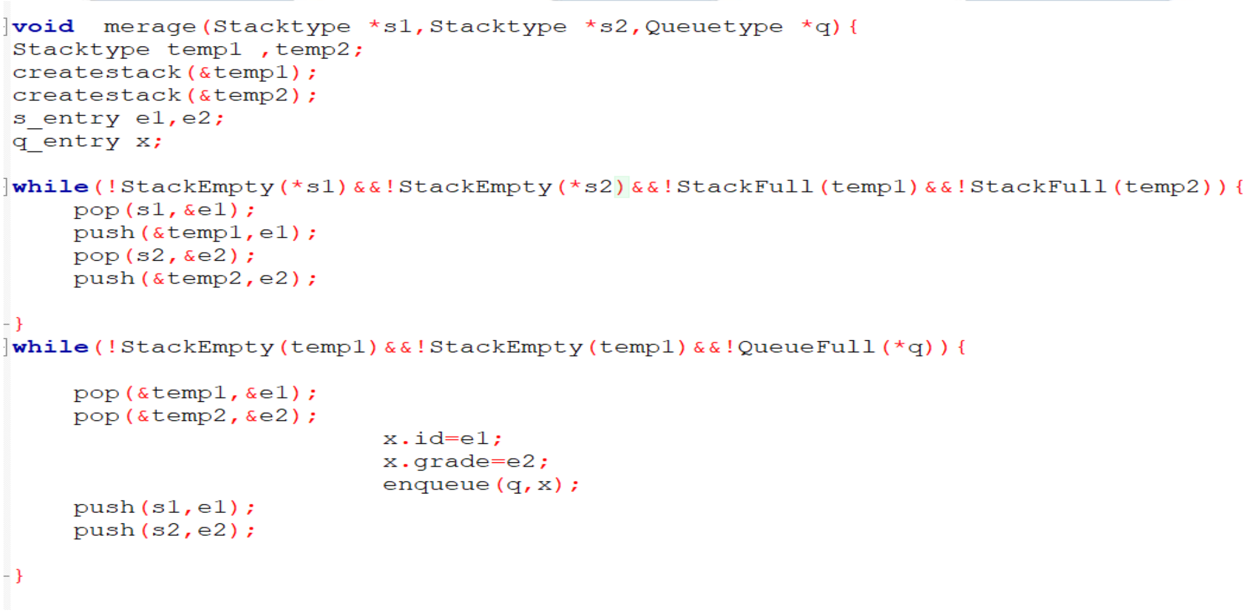
```
typedef  struct{
int id ,grade ;

}q_entry;
```

STACK .h (changes)

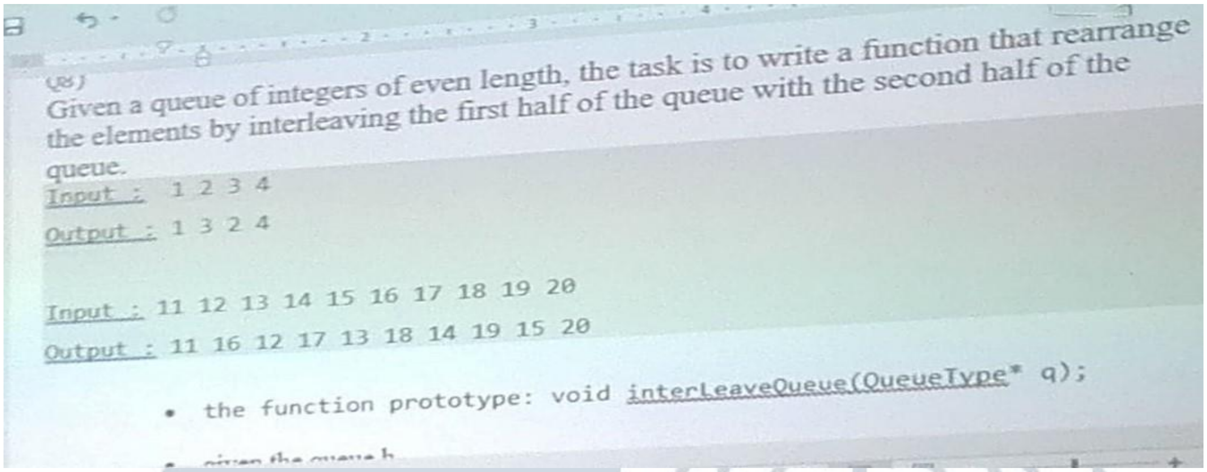


```
typedef int s_entry;
```

FUNCTION (USER LEVEL)=MAIN.C



```
void  merage(Stacktype *s1,Stacktype *s2,Queuetype *q){
Stacktype temp1 ,temp2;
createstack(&temp1);
createstack(&temp2);
s_entry e1,e2;
q_entry x;

while(!StackEmpty(*s1)&&!StackEmpty(*s2)&&!StackFull(temp1)&&!StackFull(temp2)){
    pop(s1,&e1);
    push(&temp1,e1);
    pop(s2,&e2);
    push(&temp2,e2);

}
while(!StackEmpty(temp1)&&!StackEmpty(temp1)&&!QueueFull(*q)){

    pop(&temp1,&e1);
    pop(&temp2,&e2);
                        x.id=e1;
                        x.grade=e2;
                        enqueue(q,x);
    push(s1,e1);
    push(s2,e2);

}
```



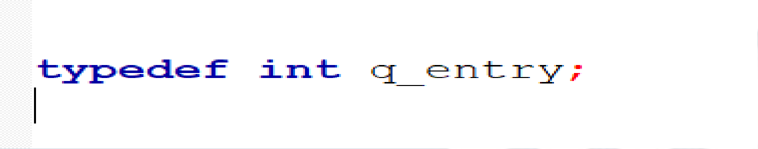
(8)
Given a queue of integers of even length, the task is to write a function that rearrange the elements by interleaving the first half of the queue with the second half of the queue.

Input : 1 2 3 4
Output : 1 3 2 4

Input : 11 12 13 14 15 16 17 18 19 20
Output : 11 16 12 17 13 18 14 19 15 20

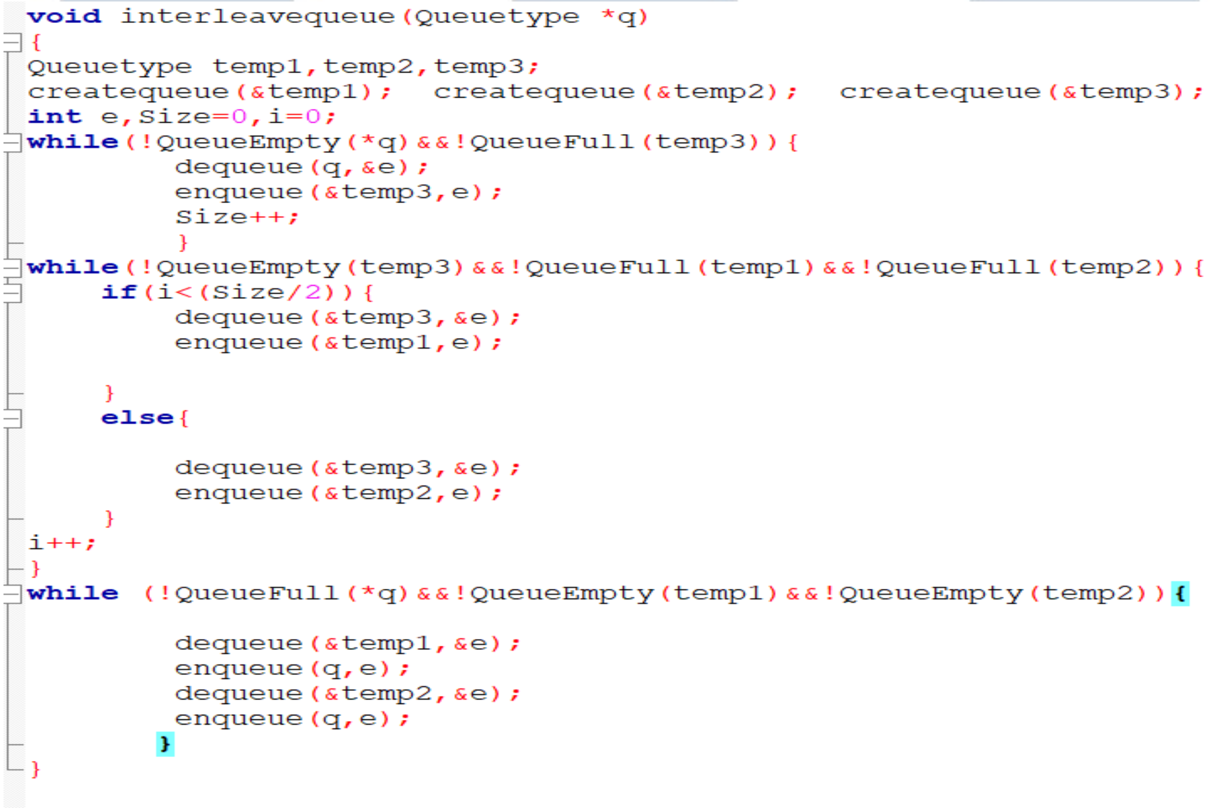
- the function prototype: void interLeaveQueue(QueueType* q);
- given the queue.h

QUEUE .h

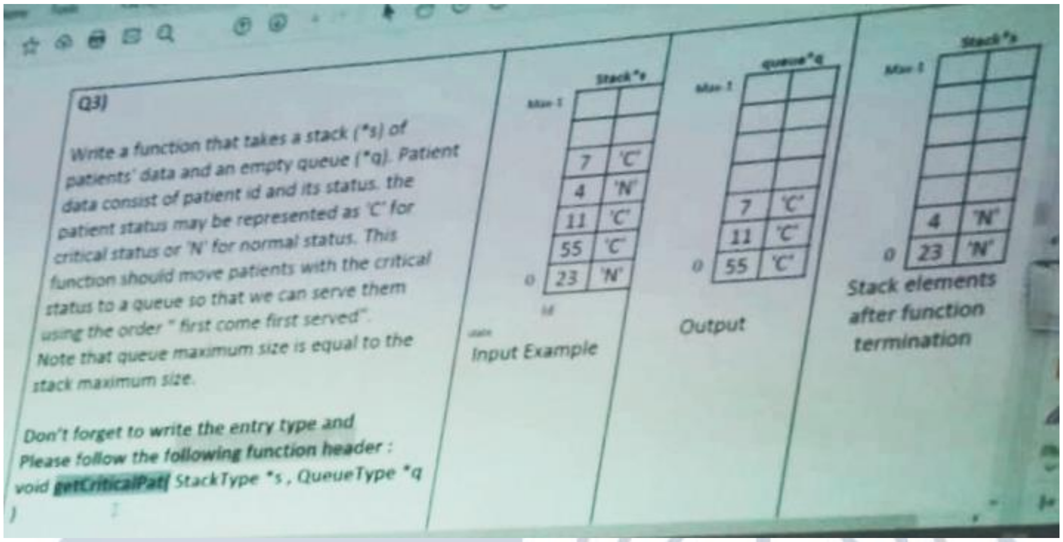


```c
typedef int q_entry;
```

FUNCTION (USER LEVEL)=MAIN.C



```c
void interleavequeue(Queuetype *q)
{
Queuetype temp1,temp2,temp3;
createqueue(&temp1);  createqueue(&temp2);  createqueue(&temp3);
int e,Size=0,i=0;
while(!QueueEmpty(*q)&&!QueueFull(temp3)){
        dequeue(q,&e);
        enqueue(&temp3,e);
        Size++;
        }
while(!QueueEmpty(temp3)&&!QueueFull(temp1)&&!QueueFull(temp2)){
    if(i<(Size/2)){
        dequeue(&temp3,&e);
        enqueue(&temp1,e);

    }
    else{

        dequeue(&temp3,&e);
        enqueue(&temp2,e);
    }
i++;
}
while  (!QueueFull(*q)&&!QueueEmpty(temp1)&&!QueueEmpty(temp2)){

        dequeue(&temp1,&e);
        enqueue(q,e);
        dequeue(&temp2,&e);
        enqueue(q,e);
       }
}
```



Q3)

Write a function that takes a stack (*s) of patients' data and an empty queue (*q). Patient data consist of patient id and its status. the patient status may be represented as 'C' for critical status or 'N' for normal status. This function should move patients with the critical status to a queue so that we can serve them using the order " first come first served".
Note that queue maximum size is equal to the stack maximum size.

Don't forget to write the entry type and
Please follow the following function header :
void getCriticalPat(StackType *s , QueueType *q
)

Input Example

	Stack *s	
Max-1		
	7	'C'
	4	'N'
	11	'C'
	55	'C'
0	23	'N'

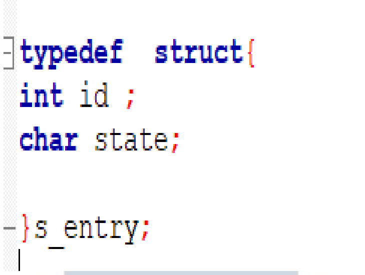
Output

	queue *q	
Max-1		
	7	'C'
	11	'C'
0	55	'C'

	Stack *s	
Max-1		
	4	'N'
0	23	'N'

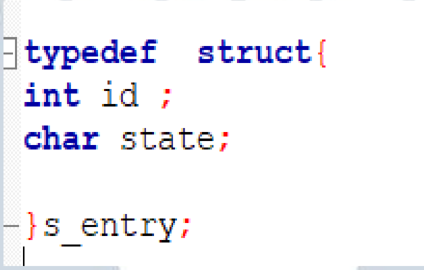
Stack elements after function termination

QUEUE .h (changes)



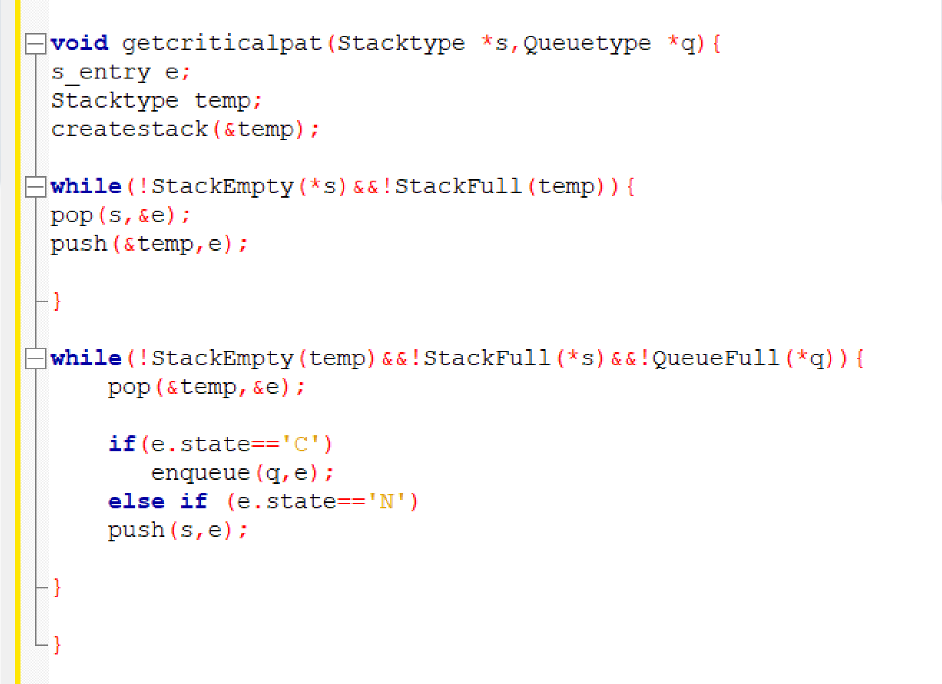
```
typedef  struct{
int id ;
char state;

}s_entry;
```

STACK .h (changes)



```
typedef  struct{
int id ;
char state;

}s_entry;
```

FUNCTION (USER LEVEL)=MAIN.C



```
void getcriticalpat(Stacktype *s,Queuetype *q){
s_entry e;
Stacktype temp;
createstack(&temp);

while(!StackEmpty(*s)&&!StackFull(temp)){
pop(s,&e);
push(&temp,e);

}

while(!StackEmpty(temp)&&!StackFull(*s)&&!QueueFull(*q)){
    pop(&temp,&e);

    if(e.state=='C')
       enqueue(q,e);
    else if (e.state=='N')
    push(s,e);

}

}
```