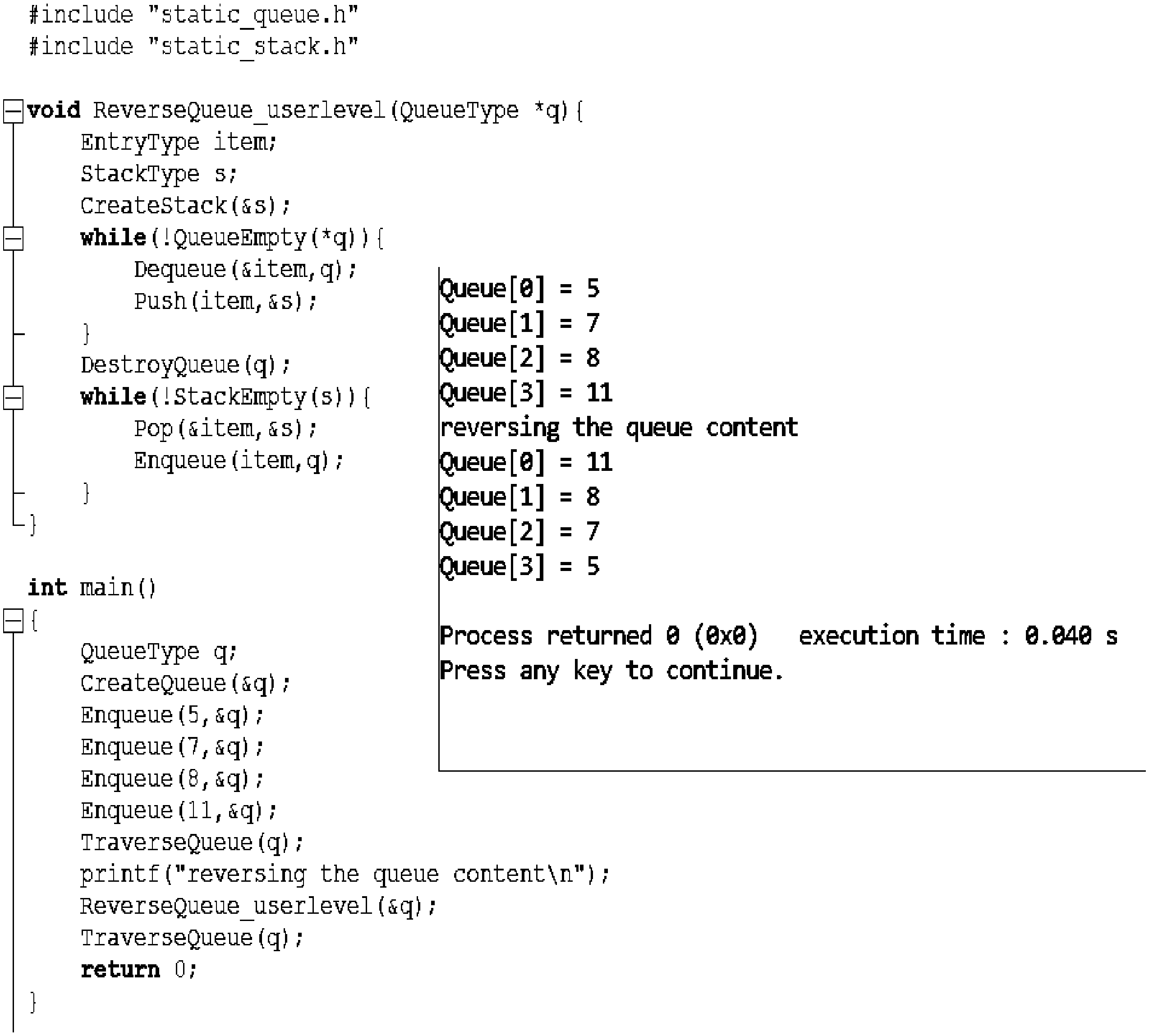
Part (1) Writing Codes

Q1. Write function that takes a queue from user and reverse its content in the same queue. [User level]

Answer



```
#include "static_queue.h"
#include "static_stack.h"

void ReverseQueue_userlevel(QueueType *q){
    EntryType item;
    StackType s;
    CreateStack(&s);
    while(!QueueEmpty(*q)){
        Dequeue(&item,q);
        Push(item,&s);
    }
    DestroyQueue(q);
    while(!StackEmpty(s)){
        Pop(&item,&s);
        Enqueue(item,q);
    }
}

int main()
{
    QueueType q;
    CreateQueue(&q);
    Enqueue(5,&q);
    Enqueue(7,&q);
    Enqueue(8,&q);
    Enqueue(11,&q);
    TraverseQueue(q);
    printf("reversing the queue content\n");
    ReverseQueue_userlevel(&q);
    TraverseQueue(q);
    return 0;
}
```

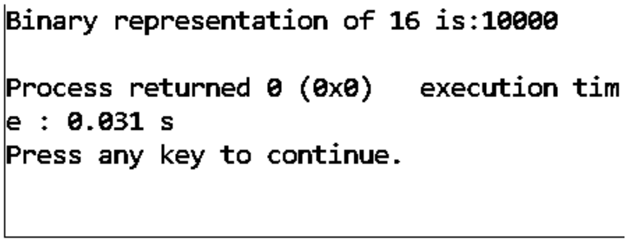
```
Queue[0] = 5
Queue[1] = 7
Queue[2] = 8
Queue[3] = 11
reversing the queue content
Queue[0] = 11
Queue[1] = 8
Queue[2] = 7
Queue[3] = 5

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

Q2. Using stacks, how to obtain the binary representation of the number? [User level]

Answer

In main.c file

```
void Decimel2Binary(unsigned int num)
{
    StackType s;
    CreateStack(&s);
    EntryType item;
    while (num != 0 && !StackFull(s))
    {
        EntryType digit = num % 2;
        Push(digit,&s);
        num = num / 2;
    }

    while (!StackEmpty(s))
    {
        Pop(&item,&s);
        printf("%d",item);
    }
}

int main()
{

    printf("Binary representation of 16 is:");
    Decimel2Binary(16);
    printf("\n");

    return 0;
}
```



```
Binary representation of 16 is:10000

Process returned 0 (0x0)   execution tim
e : 0.031 s
Press any key to continue.
```

Q3. Write a function to print on the screen the contents of a stack without changing the stack (user level)

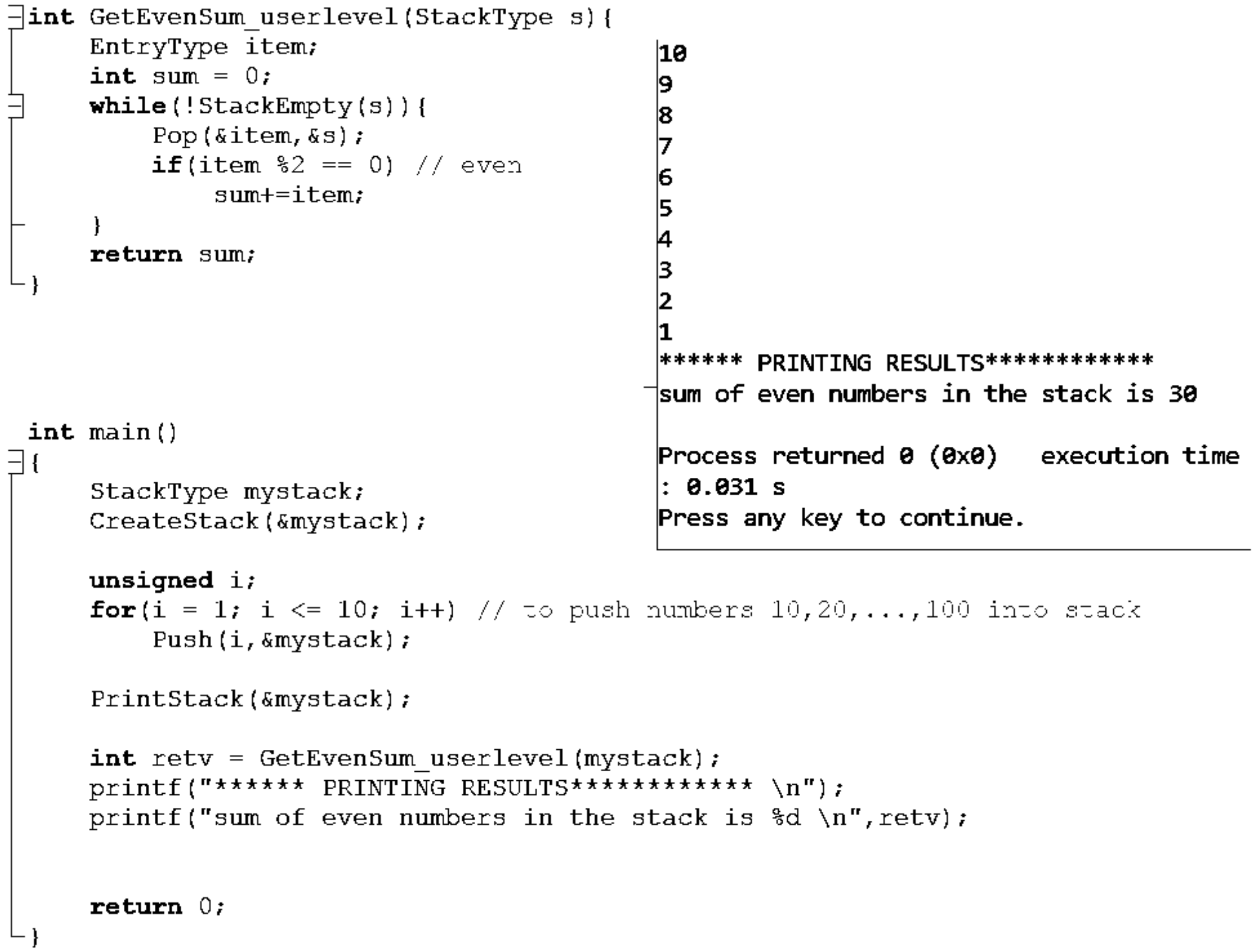
Answer

```
void PrintStack_userlevel(StackType s){
    EntryType item;
    while(!StackEmpty(s)){
        Pop(&item,&s);
        printf("%d\n",item);
    }
}
```

Q4. Write a function that returns the sum of even numbers inside a stack [user level and implementation level]

Answer

In main.c file [user]

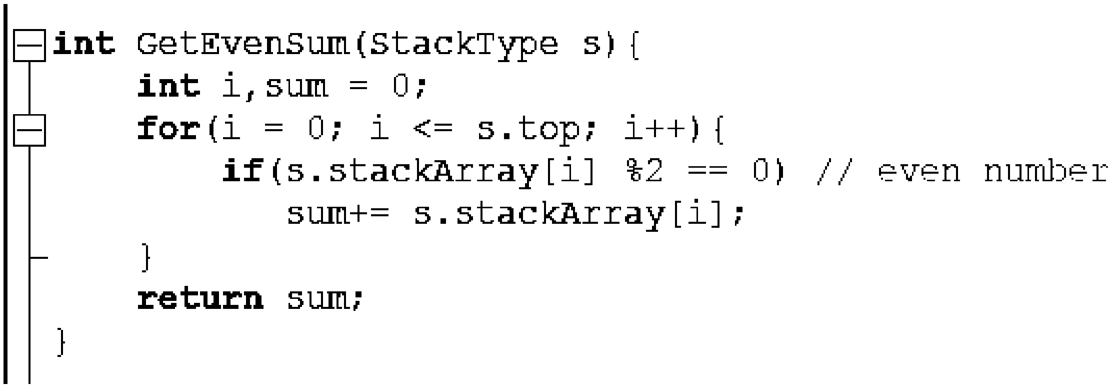


```
int GetEvenSum_userlevel(StackType s){
    EntryType item;
    int sum = 0;
    while(!StackEmpty(s)){
        Pop(&item,&s);
        if(item %2 == 0) // even
            sum+=item;
    }
    return sum;
}
```

```
10
9
8
7
6
5
4
3
2
1
****** PRINTING RESULTS************
sum of even numbers in the stack is 30

Process returned 0 (0x0)   execution time
: 0.031 s
Press any key to continue.
```

```
int main()
{
    StackType mystack;
    CreateStack(&mystack);

    unsigned i;
    for(i = 1; i <= 10; i++) // to push numbers 10,20,...,100 into stack
        Push(i,&mystack);

    PrintStack(&mystack);

    int retv = GetEvenSum_userlevel(mystack);
    printf("****** PRINTING RESULTS************ \n");
    printf("sum of even numbers in the stack is %d \n",retv);


    return 0;
}
```

In stack.c file [impl]

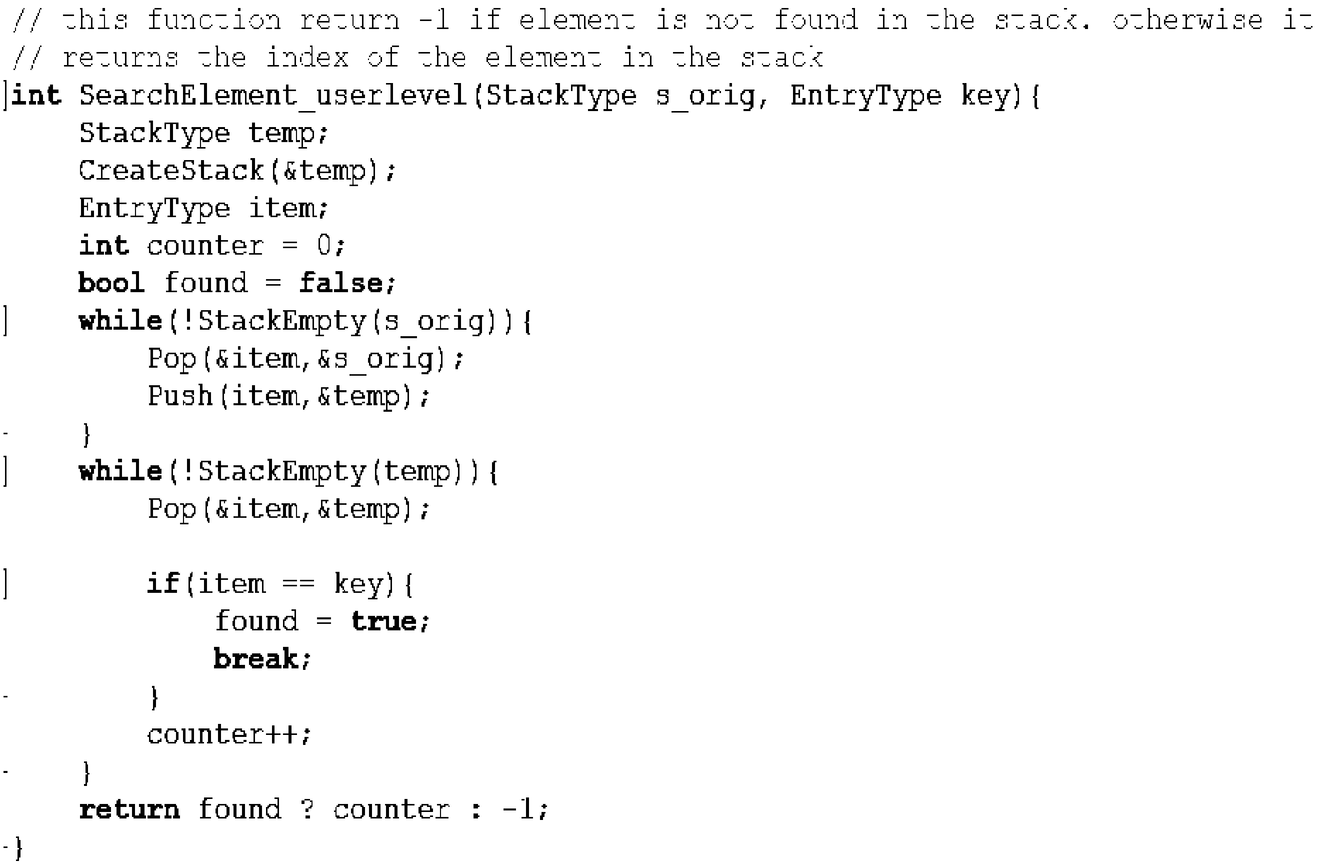


```
int GetEvenSum(StackType s){
    int i,sum = 0;
    for(i = 0; i <= s.top; i++){
        if(s.stackArray[i] %2 == 0) // even number
            sum+= s.stackArray[i];
    }
    return sum;
}
```

Q5. Write a function that searches a key inside a stack and returns its index. [User level and implementation level]

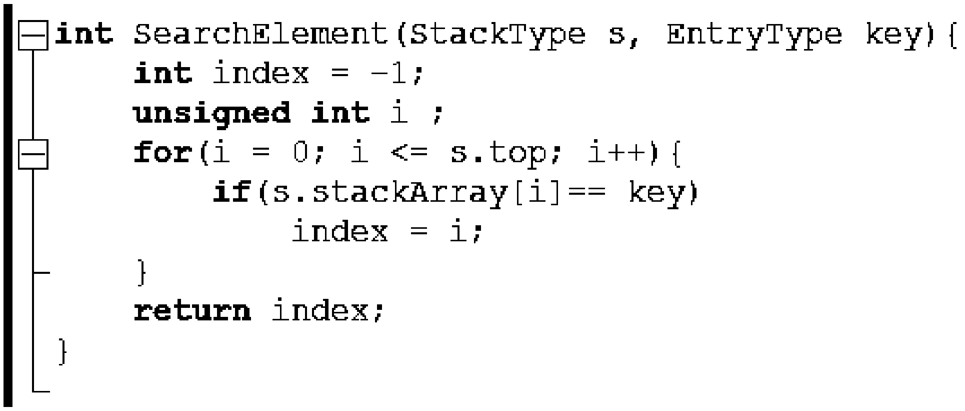
Answer

In main.c file



```
// this function return -1 if element is not found in the stack. otherwise it
// returns the index of the element in the stack
int SearchElement_userlevel(StackType s_orig, EntryType key){
    StackType temp;
    CreateStack(&temp);
    EntryType item;
    int counter = 0;
    bool found = false;
    while(!StackEmpty(s_orig)){
        Pop(&item,&s_orig);
        Push(item,&temp);
    }
    while(!StackEmpty(temp)){
        Pop(&item,&temp);

        if(item == key){
            found = true;
            break;
        }
        counter++;
    }
    return found ? counter : -1;
}
```

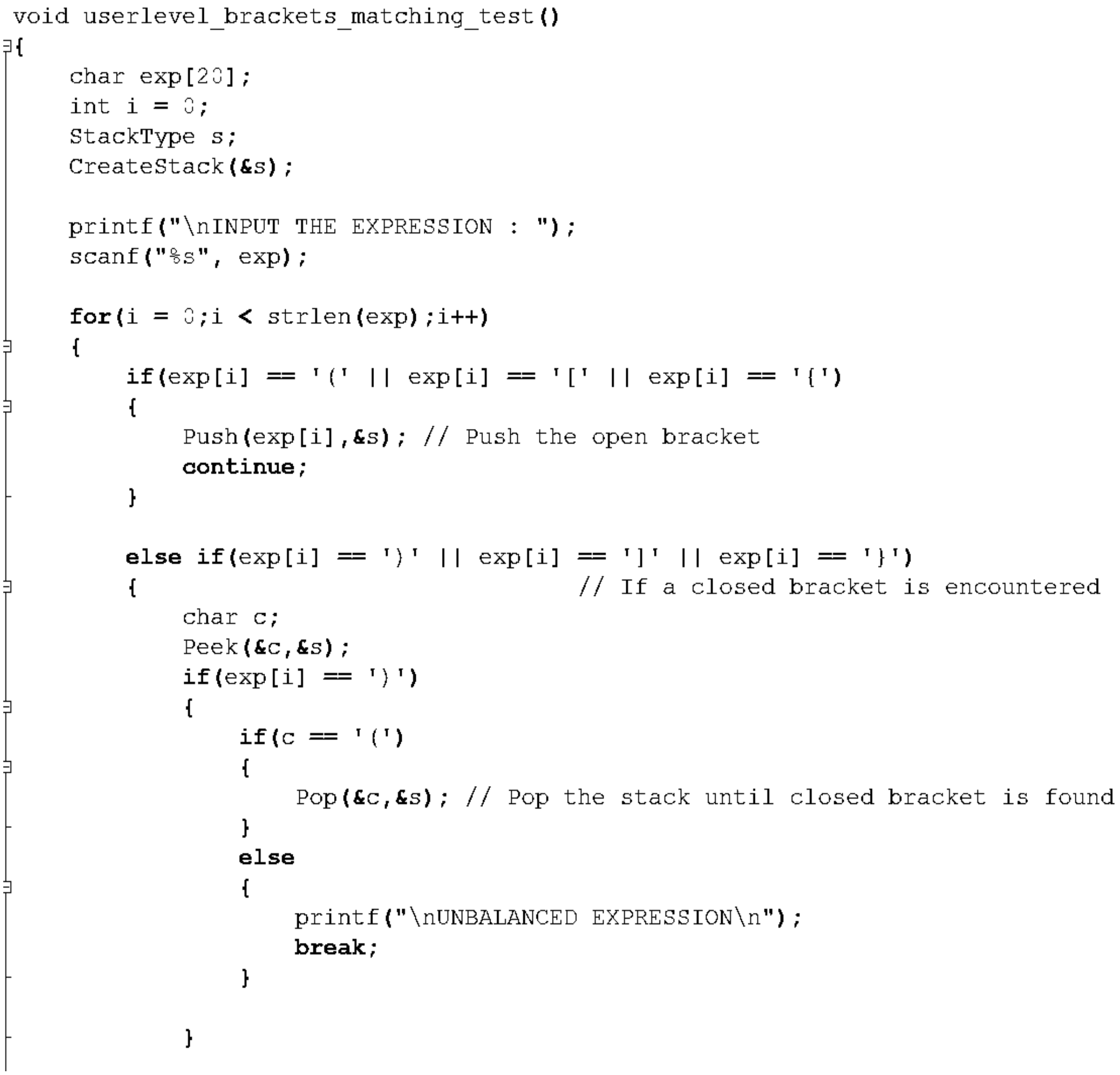
In stack.c file



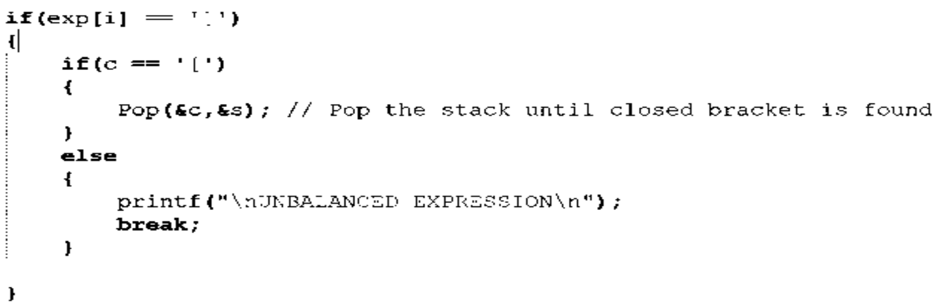
```
int SearchElement(StackType s, EntryType key){
    int index = -1;
    unsigned int i ;
    for(i = 0; i <= s.top; i++){
        if(s.stackArray[i]== key)
            index = i;
    }
    return index;
}
```

Q6. Use a stack structure to check the balance and ordering between various parentheses.

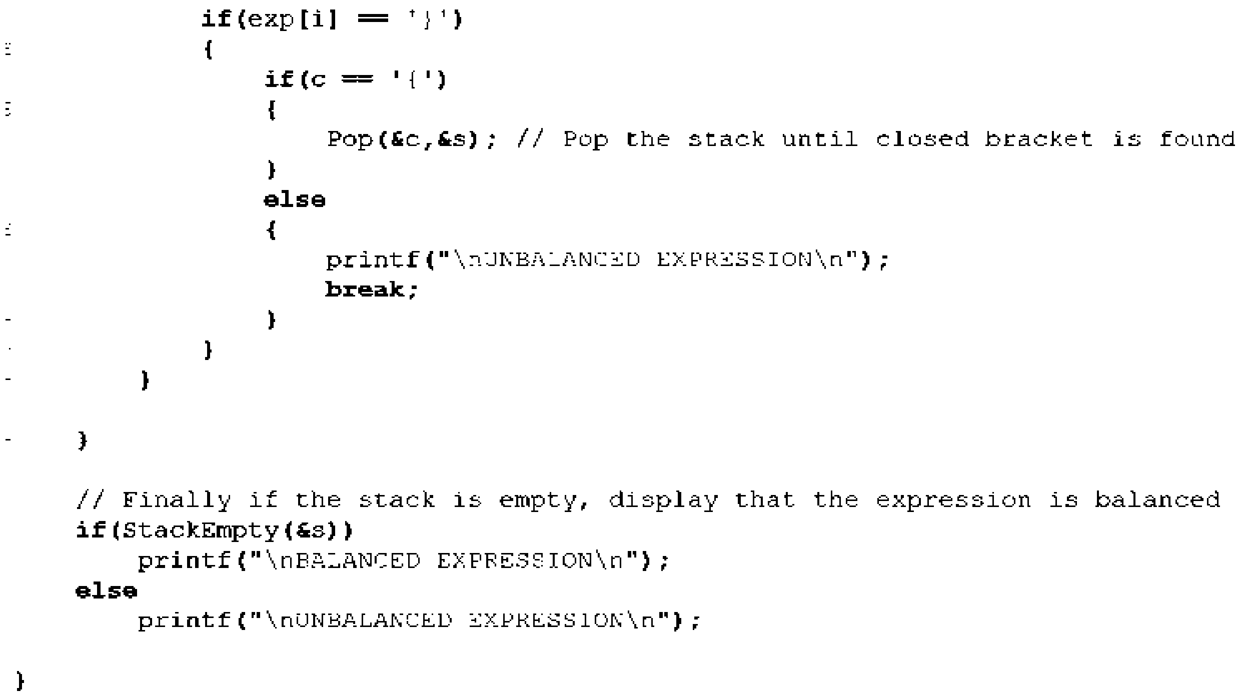
Answer



```
void userlevel_brackets_matching_test()
{
    char exp[20];
    int i = 0;
    StackType s;
    CreateStack(&s);

    printf("\nINPUT THE EXPRESSION : ");
    scanf("%s", exp);

    for(i = 0;i < strlen(exp);i++)
    {
        if(exp[i] == '(' || exp[i] == '[' || exp[i] == '{')
        {
            Push(exp[i],&s); // Push the open bracket
            continue;
        }

        else if(exp[i] == ')' || exp[i] == ']' || exp[i] == '}')
        {                               // If a closed bracket is encountered
            char c;
            Peek(&c,&s);
            if(exp[i] == ')')
            {
                if(c == '(')
                {
                    Pop(&c,&s); // Pop the stack until closed bracket is found
                }
                else
                {
                    printf("\nUNBALANCED EXPRESSION\n");
                    break;
                }

            }
```



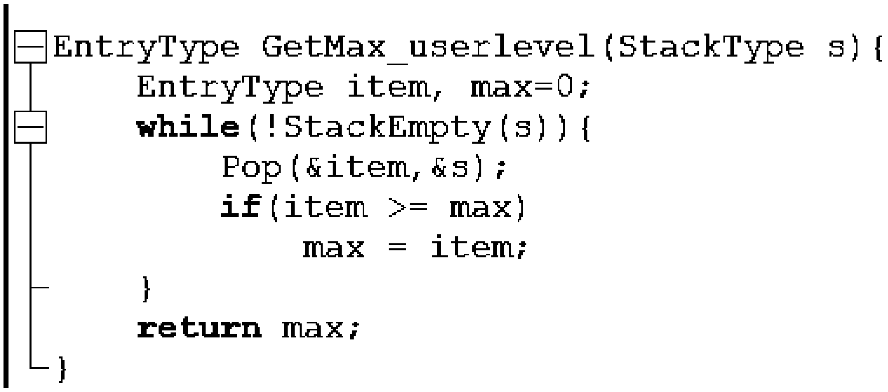
```
if(exp[i] == ']')
{
    if(c == '[')
    {
        Pop(&c,&s); // Pop the stack until closed bracket is found
    }
    else
    {
        printf("\nUNBALANCED EXPRESSION\n");
        break;
    }

}
```



```
            if(exp[i] == '}')
            {
                if(c == '{')
                {
                    Pop(&c,&s); // Pop the stack until closed bracket is found
                }
                else
                {
                    printf("\nUNBALANCED EXPRESSION\n");
                    break;
                }
            }
        }

    }

    // Finally if the stack is empty, display that the expression is balanced
    if(StackEmpty(&s))
        printf("\nBALANCED EXPRESSION\n");
    else
        printf("\nUNBALANCED EXPRESSION\n");

}
```

Q7. Write a function is to get the max number in a stack and a queue. [User level]

Answer

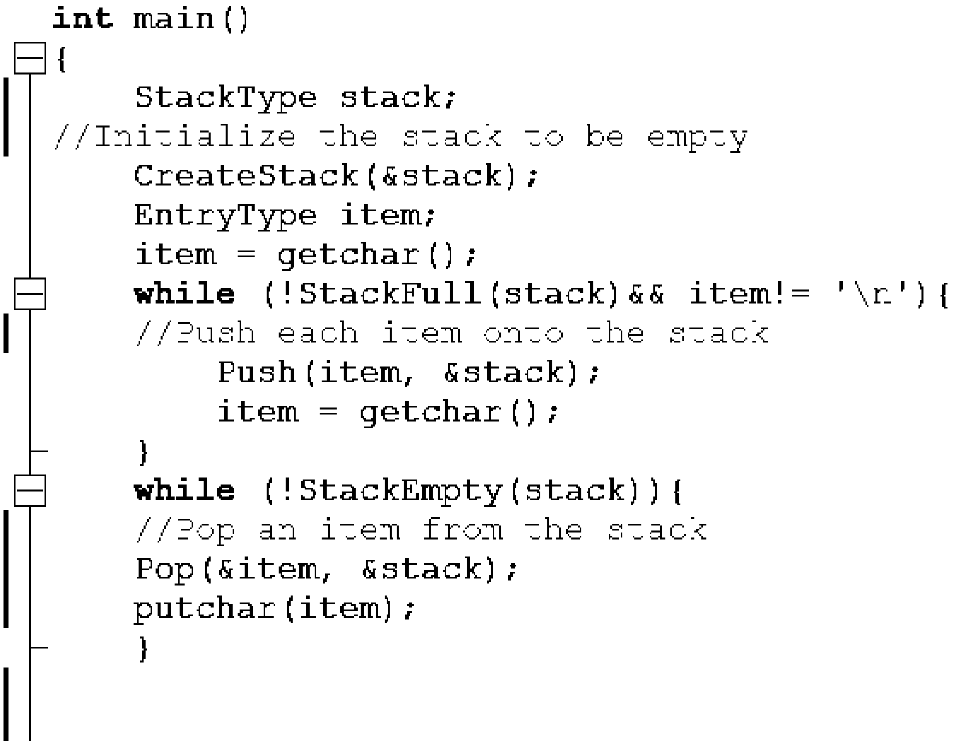


```
EntryType GetMax_userlevel(StackType s){
    EntryType item, max=0;
    while(!StackEmpty(s)){
        Pop(&item,&s);
        if(item >= max)
            max = item;
    }
    return max;
}
```

Q8. Write code to reverse a string using stack. [user and impl level]

Answer

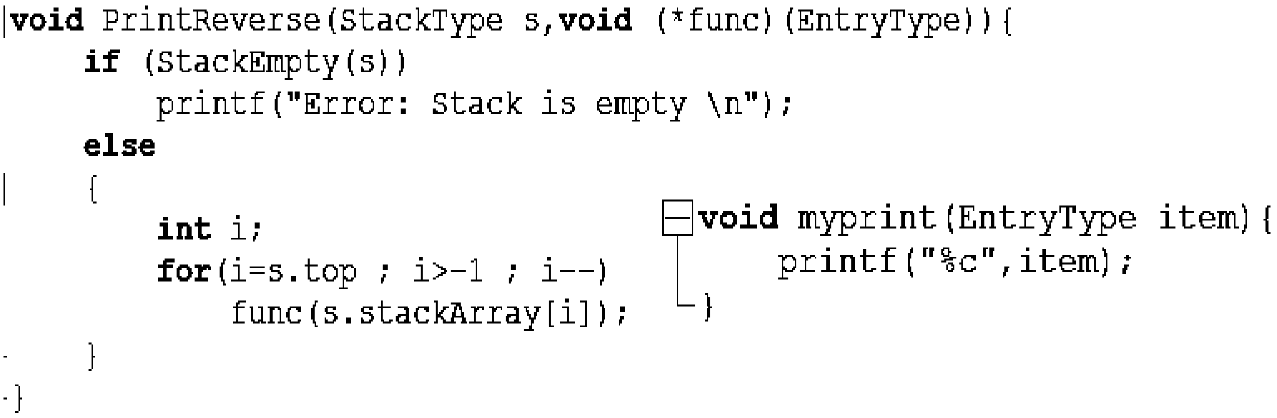
In main.c file



```
int main()
{
    StackType stack;
//Initialize the stack to be empty
    CreateStack(&stack);
    EntryType item;
    item = getchar();
    while (!StackFull(stack)&& item!= '\n'){
    //Push each item onto the stack
        Push(item, &stack);
        item = getchar();
    }
    while (!StackEmpty(stack)){
    //Pop an item from the stack
    Pop(&item, &stack);
    putchar(item);
    }
```

In stack.c file

// remember to typedef char EntryType; in stack.h



```
void PrintReverse(StackType s,void (*func)(EntryType)){
    if (StackEmpty(s))
        printf("Error: Stack is empty \n");
    else
    {
        int i;
        for(i=s.top ; i>-1 ; i--)
            func(s.stackArray[i]);
    }
}
```

```
void myprint(EntryType item){
    printf("%c",item);
}
```

Q9. How to create a queue using stacks?

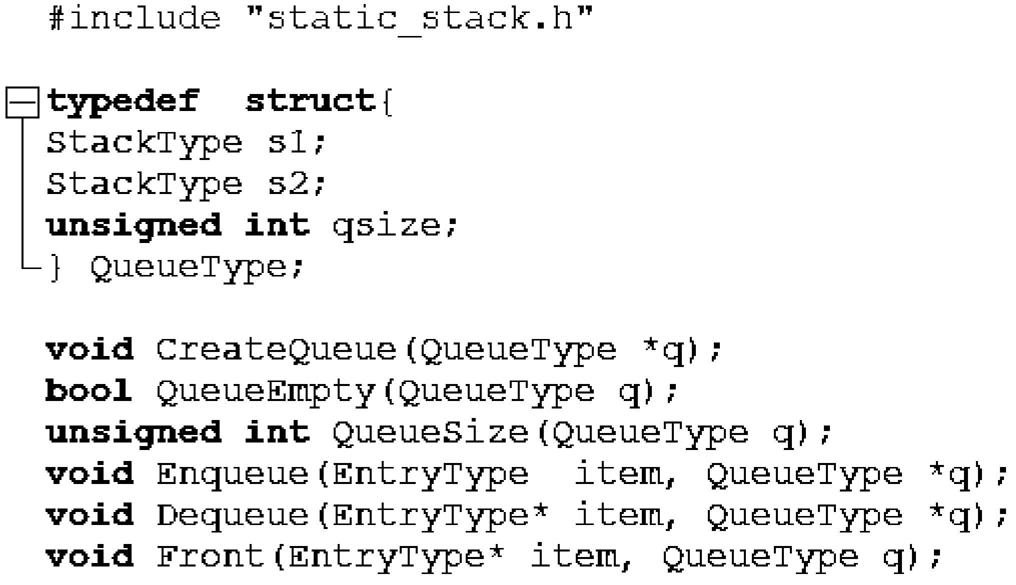
Answer

Queue could be implemented using 2 stacks.

- (By making deQueue operation costly)In this method, in en-queue operation, the new element is entered at the top of stack1. In de-queue operation, if stack2 is empty then all the elements are moved to stack2 and finally top of stack2 is returned.

```
enQueue(q,  x)
  1) Push x to stack1
deQueue(q)
  1) If both stacks are empty then error.
  2) If stack2 is empty
       While stack1 is not empty, push everything from stack1 to stack2.
  3) Pop the element from stack2 and return it.
```

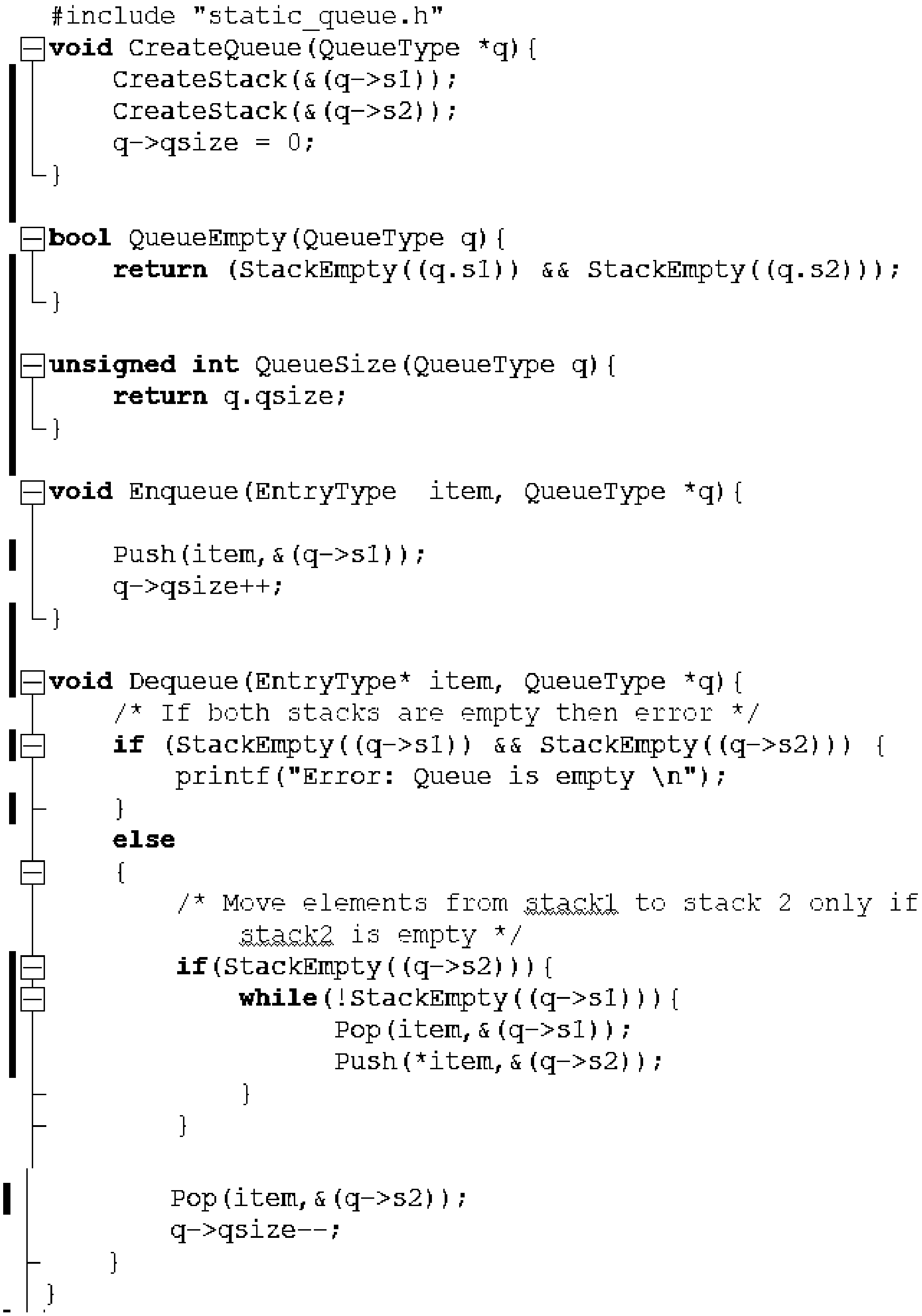
queue.h



```
#include "static_stack.h"

typedef  struct{
StackType s1;
StackType s2;
unsigned int qsize;
} QueueType;

void CreateQueue(QueueType *q);
bool QueueEmpty(QueueType q);
unsigned int QueueSize(QueueType q);
void Enqueue(EntryType  item, QueueType *q);
void Dequeue(EntryType* item, QueueType *q);
void Front(EntryType* item, QueueType q);
```

queue.c file

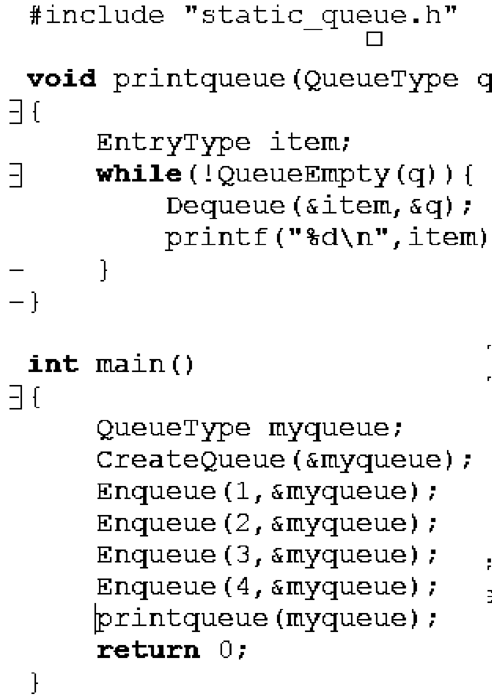
Note that we may use size to check the number of elements inside queue.



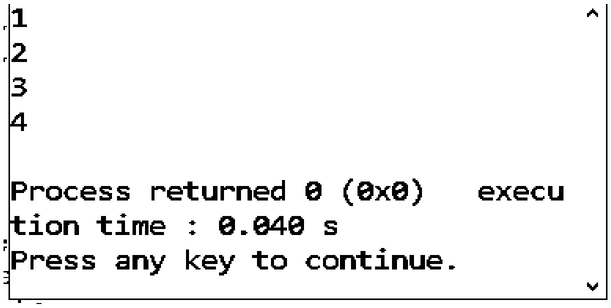
```
#include "static_queue.h"
void CreateQueue(QueueType *q){
    CreateStack(&(q->s1));
    CreateStack(&(q->s2));
    q->qsize = 0;
}

bool QueueEmpty(QueueType q){
    return (StackEmpty((q.s1)) && StackEmpty((q.s2)));
}

unsigned int QueueSize(QueueType q){
    return q.qsize;
}

void Enqueue(EntryType  item, QueueType *q){

    Push(item,&(q->s1));
    q->qsize++;
}

void Dequeue(EntryType* item, QueueType *q){
    /* If both stacks are empty then error */
    if (StackEmpty((q->s1)) && StackEmpty((q->s2))) {
        printf("Error: Queue is empty \n");
    }
    else
    {
        /* Move elements from stack1 to stack 2 only if
           stack2 is empty */
        if(StackEmpty((q->s2))){
            while(!StackEmpty((q->s1))){
                   Pop(item,&(q->s1));
                   Push(*item,&(q->s2));
            }
        }

        Pop(item,&(q->s2));
        q->qsize--;
    }
}
```

main.c file

Note the using of queue functions



```
#include "static_queue.h"

void printqueue(QueueType q)
{
    EntryType item;
    while(!QueueEmpty(q)){
        Dequeue(&item,&q);
        printf("%d\n",item);
    }
}

int main()
{
    QueueType myqueue;
    CreateQueue(&myqueue);
    Enqueue(1,&myqueue);
    Enqueue(2,&myqueue);
    Enqueue(3,&myqueue);
    Enqueue(4,&myqueue);
    printqueue(myqueue);
    return 0;
}
```

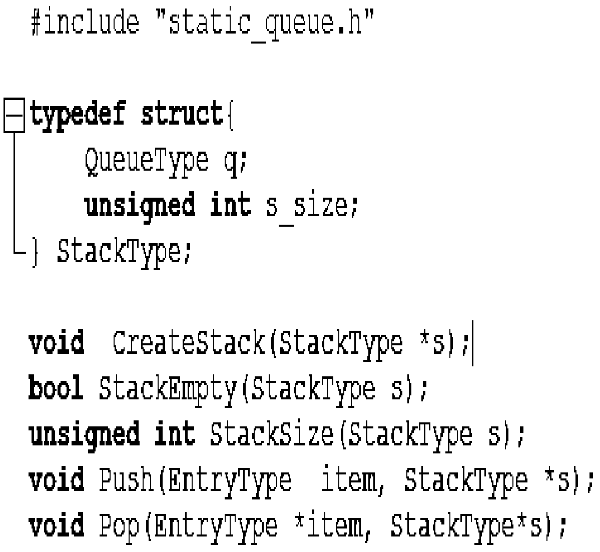


```
1
2
3
4

Process returned 0 (0x0)   execu
tion time : 0.040 s
Press any key to continue.
```

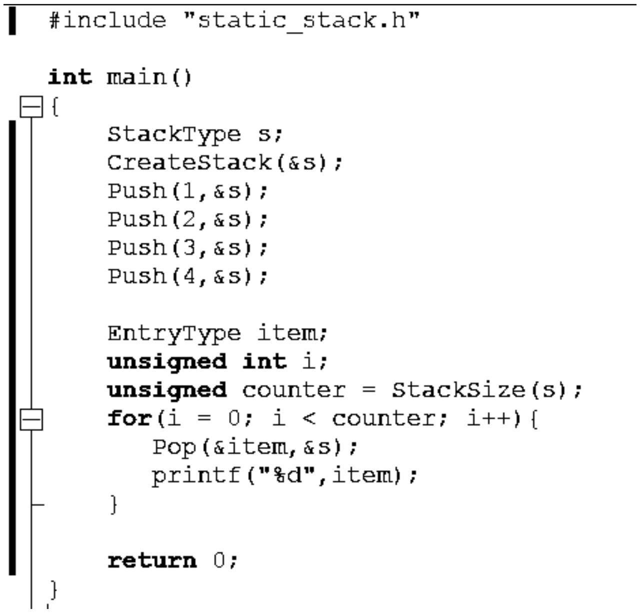
Q11. How to create a stack using queue?

Answer

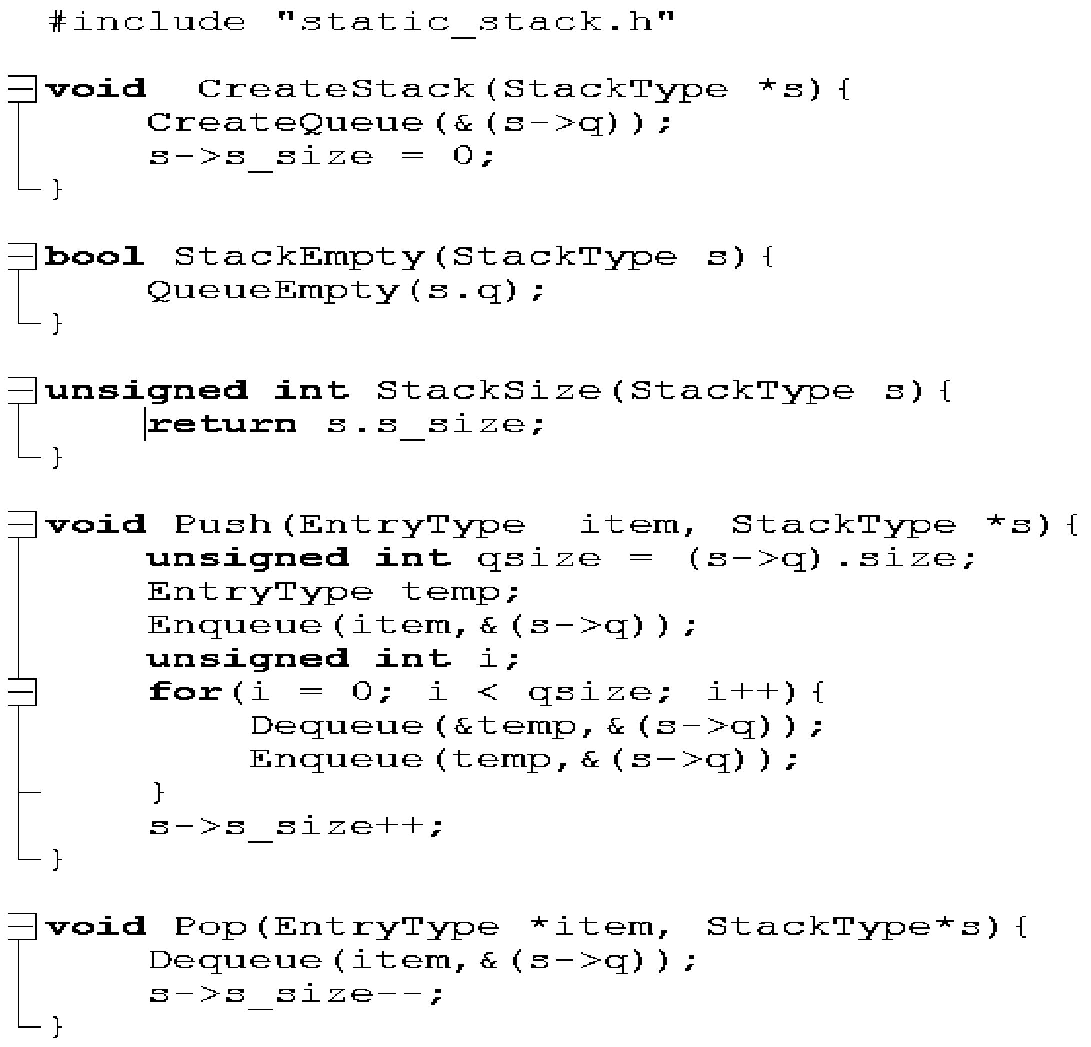
Stack could be implemented using one queue



```
#include "static_queue.h"

typedef struct{
    QueueType q;
    unsigned int s_size;
} StackType;

void  CreateStack(StackType *s);
bool StackEmpty(StackType s);
unsigned int StackSize(StackType s);
void Push(EntryType  item, StackType *s);
void Pop(EntryType *item, StackType*s);
```



```
#include "static_stack.h"

int main()
{
    StackType s;
    CreateStack(&s);
    Push(1,&s);
    Push(2,&s);
    Push(3,&s);
    Push(4,&s);

    EntryType item;
    unsigned int i;
    unsigned counter = StackSize(s);
    for(i = 0; i < counter; i++){
       Pop(&item,&s);
       printf("%d",item);
    }

    return 0;
}
```



```
#include "static_stack.h"

void  CreateStack(StackType *s){
        CreateQueue(&(s->q));
        s->s_size = 0;
}

bool StackEmpty(StackType s){
        QueueEmpty(s.q);
}

unsigned int StackSize(StackType s){
        return s.s_size;
}

void Push(EntryType  item, StackType *s){
        unsigned int qsize = (s->q).size;
        EntryType temp;
        Enqueue(item,&(s->q));
        unsigned int i;
        for(i = 0; i < qsize; i++){
                Dequeue(&temp,&(s->q));
                Enqueue(temp,&(s->q));
        }
        s->s_size++;
}

void Pop(EntryType *item, StackType*s){
        Dequeue(item,&(s->q));
        s->s_size--;
}
```

Part (2) List-array Based

List.h file

```
#include<stdio.h>

#include<conio.h>
#include<stdlib.h>
#include <stdbool.h>

#define MAX 10

typedef unsigned int Entrytype;
typedef struct {
    Entrytype listArray[MAX];
    unsigned int length;     //total no of elements
}ListType;

/************** function prototypes ****************************/
void CreateList(ListType *L);
//This function inserts the element at specified position
bool InsertAtIndex(ListType *L, Entrytype item, unsigned int
index);
//This function deletes the element at given position
bool DeleteAtIndex(ListType *L, unsigned int index);
bool ListEmpty(ListType L);
bool ListFull(ListType L);
void ListTravrse(ListType L, void (*func)(Entrytype));
bool FindElement(ListType L, Entrytype item, unsigned int* index);
```

List.c file

```
#include "staticList.h"

void CreateList(ListType *L){
    L->length = 0;
}
void ListTravrse(ListType L, void (*func)(Entrytype)){
    int i = 0;
    for(i = 0; i < L.length; i++)
        func(L.listArray[i]);
}
```

```
bool InsertAtIndex(ListType *L, Entrytype item, unsigned int
index){
    int i;
    if(index > MAX)
    {
        printf("Cannot insert at index %d because max length
is %d \n",index,MAX);
        return false;
    }

    if(index >=0 && index <= L->length)
    {
        for (i=L->length; i> index; i--)
        {
            L->listArray[i] = L->listArray[i-1];
        }

        L->listArray[index] = item;
        L->length++;
        return true;
    }
    return false; // index is out of insertion range.
}

bool DeleteAtIndex(ListType *L, unsigned int index){
    if(index > MAX)
    {
        printf("Cannot delete at index %d because max length
is %d \n",index,MAX);
        return false;
    }
    if(index >=0 && index <= (L->length-1))
    {
        int i;
        for(i = index+1; i < L->length; i++)
            L->listArray[i-1] = L->listArray[i];

        L->length--;
        return true;
    }
    return false; // index is out of deletion range.
}
```

```
bool ListEmpty(ListType L){

    return (L.length == 0);
}
bool ListFull(ListType L){
    return (L.length == MAX);
}

bool FindElement(ListType L, Entrytype item, unsigned int
*index){
    if(ListEmpty(L))
        return false; // because empty list

    unsigned int i;
    for(i = 0; i < L.length; i++){
        if(L.listArray[i] == item){
            *index = i;
            return true;
        }
    }
    return false; // item is not found
}
```

Part (3) MCQ

1. STACK IN DATA STRUCTURE IS -----

a) LILO b) FIFO c) None of these **d) LIFO**

2. PROCESS OF INSERTING AN ELEMENT IN STACK IS CALLED -----

a) Create **b) Push** c) Evaluation d) Pop

3. PROCESS OF DELETING AN ELEMENT IN STACK IS CALLED -----

a) Create b) Push c) Evaluation **d) Pop**

4. IN A STACK, IF A USER TRIES TO REMOVE AN ELEMENT FROM AN EMPTY STACK IS CALLED ---

a) Underflow b) Empty collection
c) Overflow d) Garbage collection

5. PUSHING AN ELEMENT INTO STACK ALREADY HAVING FIVE ELEMENTS AND STACK SIZE OF 5, THEN STACK BECOMES -----

a) Overflow b) Crash c) Underflow d) User flow

6. WHICH OF THE FOLLOWING ARRAY ELEMENT WILL RETURN THE TOP-OF-THE-STACK-ELEMENT FOR A STACK OF SIZE N ELEMENTS (CAPACITY OF STACK > N)?

a) S[N-1] b) S[N] c) S[N-2] d) S[N+1]

7. WHICH OF THE FOLLOWING IS NOT THE APPLICATION OF STACK?

a) A parentheses balancing program
b) tracking of local variables at run time
c) Compiler Syntax Analyzer
d) Data Transfer between two asynchronous processes

8. CONSIDER THE USUAL ALGORITHM FOR DETERMINING WHETHER A SEQUENCE OF PARENTHESES IS BALANCED. THE MAXIMUM NUMBER OF PARENTHESES THAT APPEAR ON THE STACK AT ANY ONE TIME WHEN THE ALGORITHM ANALYZES: (()(())(()))?

a) 1 b) 2 **c) 3** d) 4 or more

9. WHAT DATA STRUCTURE WOULD YOU MOSTLY LIKELY SEE IN NON-RECURSIVE IMPLEMENTATION OF A RECURSIVE ALGORITHM?

a) Linked List **b) Stack** c) Queue d) Tree

10. THE PROCESS OF ACCESSING DATA STORED IN A SERIAL ACCESS MEMORY IS SIMILAR TO MANIPULATING DATA ON A _______

a) Heap
b) Binary Tree
c) Array
d) Stack

11. WHAT IS THE RESULT OF THE FOLLOWING OPERATION?

Top (Push (S, X))

a) X
b) X+S
c) S
d) XS

12. WHICH DATA STRUCTURE IS USED FOR IMPLEMENTING RECURSION?

a) Queue
b) Stack
c) Array
d) List

13. WHICH OF THE FOLLOWING STATEMENT(S) ABOUT STACK DATA STRUCTURE IS/ARE NOT CORRECT?

a) Linked List are used for implementing Stacks
b) Top of the Stack always contain the new node
c) Stack is the FIFO data structure
d) Null link is present in the last node at the bottom of the stack

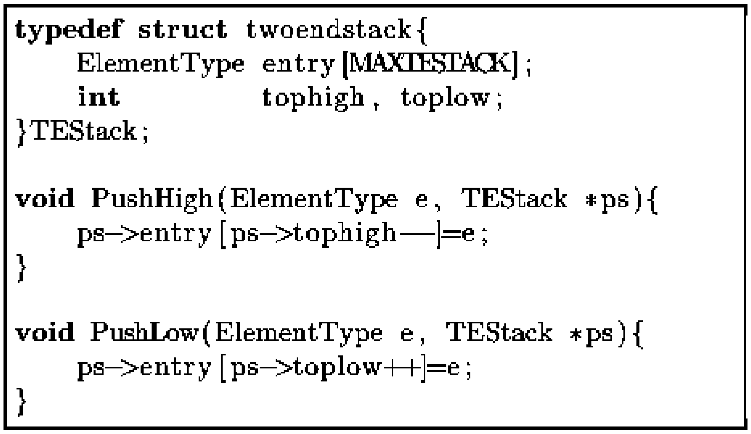
14. VOID FUN(INT N)

```
{
Stack S; // Say it creates an empty stack S
while (n > 0)
{
push(&S, n%2);
n = n/2;
}
while (!isEmpty(&S))
printf("%d ", pop(&S)); // pop an element from S and print it
}
```

What does the above function do in general?

a) Prints binary representation of n in reverse order
b) Prints binary representation of n
c) Prints the value of Logn
d) Prints the value of Logn in reverse order

15.



```
typedef struct twoendstack{
     ElementType entry[MAXTESTACK];
     int          tophigh, toplow;
}TEStack;

void PushHigh(ElementType e, TEStack *ps){
     ps->entry[ps->tophigh--]=e;
}

void PushLow(ElementType e, TEStack *ps){
     ps->entry[ps->toplow++]=e;
}
```

Code 1. Notice that, for this structure, a new element should *not* overwrite an existing one.

The body of CreateTEStack(TEStack *ps) that is written for the structure of Code 1 should be

a) ps->toplow=0; ps->tophigh=MAXTESTACK-1;

b) ps->toplow=0; ps->tophigh=MAXTESTACK;

c) ps->toplow=-1; ps->tophigh=MAXTESTACK-1;

d) ps->toplow=-1; ps->tophigh=MAXTESTACK;

16. THE FUNCTION STACKSIZE(TESTACK *PS) THAT IS WRITTEN FOR THE STRUCTURE OF CODE 1 SHOULD RETURN

a) MAXTESTACK - ps->tophigh + ps->toplow - 1;

b) MAXTESTACK - ps->tophigh - ps->toplow + 1;

c) MAXTESTACK + ps->tophigh - ps->toplow - 1;

d) ps->tophigh - ps->toplow;

17. CONSIDER THE FOLLOWING PSEUDOCODE THAT USES A STACK DECLARE A STACK OF CHARACTERS WHILE (THERE ARE MORE CHARACTERS IN THE WORD TO READ)

```
{
read a character
push the character on the stack
}
while ( the stack is not empty )
{
pop a character off the stack
write the character to the screen
}
```

What is output for input "geeksquiz"?

a) geeksquizgeeksquiz

b) ziuqskeeg

c) geeksquiz

d) ziuqskeegziuqskeeg

18. DECLARE A CHARACTER STACK WHILE (MORE INPUT IS AVAILABLE)

```
{
read a character
if ( the character is a '(' )
push it on the stack
else if ( the character is a ')' and the stack is not empty )
pop a character off the stack
else
print "unbalanced" and exit
}
print "balanced"
```

Which of these unbalanced sequences does the above code think is balanced?

a) ((())

b) ())(()

c) (()()))

d) (()))()

19. IN ARRAY-BASED IMPLEMENTATION OF STACKS, IF THE BODY OF CREATESTACK IS PS->TOP=-1, WHERE PS IS A POINTER TO THE STACK, THEN THE BODY OF PUSH SHOULD BE

a) ps->entry[ps->top++]=e;

b) ps->entry[++ps->top]=e;

c) ps->entry[--ps->top]=e;

d) ps->entry[ps->top--]=e;

20. WHAT DOES THE FOLLOWING FUNCTION CHECK FOR? (ALL NECESSARY HEADERS TO BE INCLUDED AND FUNCTION IS CALLED FROM MAIN).

```
#define MAX 10

typedef struct stack
{
    int top;
    int item[MAX];
}stack;

int function(stack *s)
{
    if(s->top == -1)
        return 1;
    else return 0;
}
```

a) Full stack b) Invalid index **c) Empty stack** d) Infinite stack

1. QUEUE IN DATA STRUCTURE IS -----

a) FIFO b) LIFO c) Ordered array d) Linear tree

2. PROCESS OF INSERTING AN ELEMENT IN QUEUE IS CALLED -----

a) Create b) Push **c) Enqueue** d) Dequeue

3. PROCESS OF REMOVING AN ELEMENT IN QUEUE IS CALLED -----

a) Enqueue b) Push c) Evaluation **d) Dequeue**

4. A LINEAR LIST OF ELEMENTS IN WHICH DELETION CAN BE DONE FROM ONE END (FRONT) AND INSERTION CAN TAKE PLACE ONLY AT THE OTHER END (REAR) IS KNOWN AS -----

a) Queue b) Stack c) Tree d) Linked list

5. IF THE ELEMENTS "A", "B", "C" AND "D" ARE PLACED IN A QUEUE AND ARE DELETED ONE AT A TIME, IN WHAT ORDER WILL THEY BE REMOVED?

a) ABCD b) DCBA c) DCAB d) ABDC

6. CIRCULAR QUEUE IS ALSO KNOWN AS -----

a) Ring Buffer b) Square Buffer
c) Rectangle Buffer d) Curve Buffer

7. WHAT DOES THE FOLLOWING CODE DO?

```
function (){
    if(isEmpty())
    return -999;
    else
    {
        entrytype high;
        high = q[front];
        return high;
    }
}
```

a) Dequeue b) Enqueue
c) Return the front element d) Return the last element

8. CONSIDER YOU HAVE A STACK WHOSE ELEMENTS IN IT ARE AS FOLLOWS.

5 4 3 2 << top Where the top element is 2.
You need to get the following stack 6 5 4 3 2 << top

THE NEEDED OPERATIONS TO BE PERFORMED ARE (YOU CAN PERFORM ONLY PUSH AND POP):

a) Push(pop()), push(6), push(pop())
b) Push(pop()), push(6)
c) Push(pop()), push(pop()), push(6)

d) Push(6)

9. A NORMAL QUEUE, IF IMPLEMENTED USING AN ARRAY OF SIZE MAX_SIZE, GETS FULL WHEN?

a) Rear = MAX_SIZE – 1

b) Front = (rear + 1) % MAX_SIZE

c) Front = rear + 1 d) Rear = front

10. IF THE MAX_SIZE IS THE SIZE OF THE ARRAY USED IN THE IMPLEMENTATION OF CIRCULAR QUEUE. HOW IS REAR MANIPULATED WHILE INSERTING AN ELEMENT IN THE QUEUE?

a) rear = (rear % 1) + MAX_SIZE

b) rear = rear % (MAX_SIZE + 1)

c) rear = (rear + 1) % MAX_SIZE

d) rear = rear + (1 % MAX_SIZE)

11. IF THE MAX_SIZE IS THE SIZE OF THE ARRAY USED IN THE IMPLEMENTATION OF CIRCULAR QUEUE, ARRAY INDEX STARTS WITH 0, FRONT POINT TO THE FIRST ELEMENT IN THE QUEUE, AND REAR POINT TO THE LAST ELEMENT IN THE QUEUE. WHICH OF THE FOLLOWING CONDITION SPECIFY THAT CIRCULAR QUEUE IS FULL?

a) Front = rear = -1

b) Front = (rear + 1) % MAX_SIZE

c) Rear = front + 1 d) Rear = (front + 1) % MAX_SIZE

12. IF THE MAX_SIZE IS THE SIZE OF THE ARRAY USED IN THE IMPLEMENTATION OF CIRCULAR QUEUE, ARRAY INDEX STARTS WITH 0, FRONT POINT TO THE FIRST ELEMENT IN THE QUEUE, AND REAR POINT TO THE LAST ELEMENT IN THE QUEUE. WHICH OF THE FOLLOWING CONDITION SPECIFY THAT CIRCULAR QUEUE IS EMPTY?

a) Front = rear = 0 **b) Front = rear = -1**

c) Front = rear + 1 d) Front = (rear + 1) % MAX_SIZE

13. QUEUES SERVE MAJOR ROLE IN -----

a) Simulation of recursion

b) Simulation of arbitrary linked list

c) Simulation of limited resource allocation

d) Simulation of heap sort

14. FOLLOWING IS C LIKE PSEUDO CODE OF A FUNCTION THAT TAKES A QUEUE AS AN ARGUMENT AND USES A STACK S TO DO PROCESSING.

```
void fun(Queue *Q)
{
    Stack S;
    while (!isEmpty(Q))
    {
        push(&S, deQueue(Q));
    }
    while (!isEmpty(&S))
    {
        enQueue(Q, pop(&S));
    }
}
```

What does the above function do in general?

a) Remove the last from Q

b) Keeps the Q same as it was before the call

c) Makes Q Empty **d) Reverses the Q**

15. WHICH ONE OF THE FOLLOWING IS AN APPLICATION OF QUEUE DATA STRUCTURE?

a) When a resource is shared among multiple consumers.

b) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes

c) Computer processes

d) All of the above

16. HOW MANY STACKS ARE NEEDED TO IMPLEMENT A QUEUE? CONSIDER THE SITUATION WHERE NO OTHER DATA STRUCTURE LIKE ARRAYS, LINKED LIST IS AVAILABLE TO YOU.

a) 1 **b) 2** c) 3 d) 4

17. A CIRCULAR QUEUE IS IMPLEMENTED USING AN ARRAY OF SIZE 10. THE ARRAY INDEX STARTS WITH 0, FRONT IS 6, AND REAR IS 9. THE INSERTION OF NEXT ELEMENT TAKES PLACE AT THE ARRAY INDEX.

a) 0 b) 7 c) 9 d) 10

أنتهت المراجعة