

Section C – Stack and Queues [Answer 1 Specified Qn from this Section]

Information: Program templates for questions 1-7 are given as separated files. You must use them to implement your functions.

1. **(createQueueFromLinkedList)** Write a C function `createQueueFromLinkedList()` to create a queue (linked-list-based) by enqueueing all integers which are stored in the linked list. The first node of the linked list is enqueued first, and then the second node, and so on. Remember to empty the queue at the beginning, if the queue is not empty.

After the queue is built, write another C function `removeOddValues()` to remove all odd integers from the queue. Note that you should only use `enqueue()` or `dequeue()` when you add or remove integers from queues.

The function prototypes are given as follows:

```
void createQueueFromLinkedList(LinkedList *ll ,  
Queue *q); void removeOddValues(Queue *q)
```

A sample input and output session is given below (if the current linked list is 1 2 3 4 5):

```
The resulting linked list is: 1 2 3 4 5
```

```
Please input your choice(1/2/3/0): 2
```

```
The resulting queue is: 1 2 3 4 5
```

```
Please input your choice(1/2/3/0): 3
```

```
The resulting queue after removing odd integers is: 2 4
```

2. **(createStackFromLinkedList)** Write a C function `createStackFromLinkedList()` to create a stack (linked-list-based) by pushing all integers that are storing in the linked list. The first node of the linked list is pushed first, and then the second node, and so on. Remember to empty the stack at the beginning, if the stack is not empty.

After the stack is built, write another C function `removeEvenValues()` to remove all even integers from the stack. Note that you should **only** use `push()` or `pop()` when you add or remove integers from the stack.

The function prototypes are given as follows:

```
void createStackFromLinkedList(LinkedList *ll , Stack  
*stack);  
void removeEvenValues(Stack *s);
```

A sample input and output session is given below (if the current linked list is 1 3 5 6 7):

```
The resulting linked list is: 1 3 5 6 7
```

```
Please input your choice(1/2/3/0): 2
```

```
The resulting stack is: 7 6 5 3 1
```

```
Please input your choice(1/2/3/0): 3
```

```
The resulting stack after removing even integers is: 7 5 3 1
```

3. **(isStackPairwiseConsecutive)** Write a C function write a function `isStackPairwiseConsecutive()` that checks whether numbers in the stack are pairwise consecutive or not. Note that the `isStackPairwiseConsecutive()` function **only** uses `push()` and `pop()` when adding or removing integers from the stack.

The function prototype is given as follows:

```
int isStackPairwiseConsecutive(Stack *s);
```

Sample test cases are given below:

For example, if the stack is 16 15 11 10 5 4:

The stack is: 16 15 11 10 5 4

The stack is pairwise consecutive

For example, if the stack is 16 15 11 10 5 1

The stack is: 16 15 11 10 5 1

The stack is not pairwise consecutive

For example, if the stack is 16 15 11 10 5

The stack is: 16 15 11 10 5

The stack is not pairwise consecutive

A sample input and output session is given below (if the current stack is 16 15 11 10 5 4):

1: Insert an integer into the stack:

2: Check the stack is pairwise consecutive:

0: Quit:

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 4

The stack is: 4

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 5

The stack is: 5 4

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 10

The stack is: 10 5 4

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 11

The stack is: 11 10 5 4

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 15

The stack is: 15 11 10 5 4

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 16

The stack is: 16 15 11 10 5 4

Please input your choice(1/2/0): 2

The stack is pairwise consecutive.

4. **(reverse)** Write a C function `reverse()` to reverse a queue using a stack. Note that the `reverse()` function only uses `push()` and `pop()` when adding or removing integers from the stack and only uses `enqueue()` and `dequeue()` when adding or removing integers from the queue.

The function prototype is given as follows:

```
void reverse(Queue *q);
```

For example, if the queue is <1, 2, 3, 4>, the resulting queue will be <4, 3, 2, 1>.

A sample input and output session is given below (if the current queue is 1 2 3 4):

1: Insert an integer into the queue;
2: Reverse the queue;
0: Quit;

```
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the queue: 1
The queue is: 1
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the queue: 2
The queue is: 1 2
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the queue: 3
The queue is: 1 2 3
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the queue: 4
The queue is: 1 2 3 4

Please input your choice(1/2/0): 2
The resulting queue after reversing its elements is: 4 3 2 1
```

5. **(recursiveReverse)** Write a recursive C function `recursiveReverse()` that reverses the order of items stored in a queue of integers. The prototype for the `recursiveReverse()` function is given below:

The function prototype is given as follows:

```
void recursiveReverse(Queue *q);
```

For example, if the queue is <1, 2, 3, 4, 5>, the resulting queue will be <5, 4, 3, 2, 1>.

A sample input and output session is given below:

1: Insert an integer into the Queue:
2: Recursively reverse the queue;
0: Quit:

```
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the queue: 1
The resulting queue is: 1
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the queue: 2
The resulting queue is: 1 2
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the queue: 3
The resulting queue is: 1 2 3
Please input your choice(1/2/0): 1
Input an integer that you want to insert into the queue: 4
The resulting queue is: 1 2 3 4
Please input your choice(1/2/0): 1

Input an integer that you want to insert into the queue: 5
The resulting queue is: 1 2 3 4 5

Please input your choice(1/2/0): 2
The resulting reversed queue is: 5 4 3 2 1

Please input your choice(1/2/0): 0
```

6. **(removeUntil)** Write a C function `removeUntil()` that pops all values off a stack of integers down to but not including the first occurrence of the chosen value.

The function prototype is given as follows:

```
void removeUntil(Stack *s, int value);
```

Given a stack [1 2 3 4 5 6 7] with the topmost number displayed on the left, calling `removeUntil()` with `value = 4` will produce the stack [4 5 6 7].

A sample input and output session is given below:

1: Insert an integer into the stack:

3: Remove values until the given value;

0: Quit:

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 7

The resulting stack is: 7

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 6

The resulting stack is: 6 7

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 5

The resulting stack is: 5 6 7

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 4

The resulting stack is: 4 5 6 7

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 3

The resulting stack is: 3 4 5 6 7

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 2

The resulting stack is: 2 3 4 5 6 7

Please input your choice(1/2/0): 1

Input an integer that you want to insert into the stack: 1

The resulting stack is: 1 2 3 4 5 6 7

Please input your choice(1/2/0): 2

Enter an integer value in stack to remove values until that value: 4

The resulting stack after removing values until the given value: 4 5 6 7

Please input your choice(1/2/0): 0

7. **(balanced)** Write a C function `balanced()` that determines if an expression comprised of the characters `()[]{}` is balanced. The prototype for the `balanced()` function is given below:

The function prototype is given as follows:

```
int balanced(char *expression);
```

For example, the following expressions are balanced because the order and quantity of the parentheses match:

`()`

`([])`

`{[]() []}`

A sample input and output session is given below:

```
1: Enter a string:
2: Check whether expressions comprised of the characters ()[]{}
   is balanced:
0: Quit:
```

```
Please input your choice(1/0): 1
Enter expressions without spaces to check whether it is
balanced or not: {[[]() []}
{[[]() []}
balanced!
```

```
Please input your choice(1/0): 0
```

The following expressions are not balanced:

```
{()}
[({})]
```

A sample input and output session is given below:

```
1: Enter a string:
2: Check whether expressions comprised of the characters ()[]{}
   is balanced:
0: Quit:
```

```
Please input your choice(1/0): 1
Enter expressions without spaces to check whether it is
balanced or not: [({})]
[({})]
not balanced!
```

```
Please input your choice(1/0): 0
```