**Assignment**

Q1\

لنقم بتتبع الشيفرة خطوة بخطوة:

1. تعريف متغيرات الصف والعناصر:

- تعريف صف الأرقام queue باستخدام `ArrayQueue` دون تحديد النوع، لذا سنفترض أنها `ArrayQueue<Integer>`.

- تعريف المتغيرات x و y وتعيينهما قيم 4 و 5 على التوالي.

2. إضافة العناصر إلى الصف:

- إضافة قيمة x (4) إلى الصف باستخدام `queue.enqueue(x)`.

- إضافة قيمة y (5) إلى الصف باستخدام `queue.enqueue(y)`.

3. قراءة العنصر الأمامي وإزالته من الصف:

- قراءة العنصر الأمامي من الصف وتخزينه في المتغير x باستخدام `x = queue.front()`.

- إزالة العنصر الأمامي من الصف باستخدام `queue.dequeue()`.

4. إضافة العناصر إلى الصف:

- إضافة قيمة (x + 5) إلى الصف باستخدام `queue.enqueue(x + 5)`.

- إضافة القيمة 16 إلى الصف باستخدام `queue.enqueue(16)`.

- إضافة قيمة x إلى الصف باستخدام `queue.enqueue(x)`.

- إضافة قيمة (y - 3) إلى الصف باستخدام `queue.enqueue(y - 3)`.

5. طباعة عناصر الصف:

- طباعة رسالة "Queue Elements: " باستخدام `System.out.println("Queue Elements: ")`.

- بدء حلقة while لطباعة العناصر المتبقية في الصف:

- التحقق مما إذا كان الصف فارغًا أم لا باستخدام `!queue.isEmptyQueue()`.

- طباعة العنصر الأمامي للصف باستخدام `System.out.println(queue.front())`.

- إزالة العنصر الأمامي من الصف باستخدام `queue.dequeue()`.

الآن دعونا نحلل الشفرة خطوة بخطوة:

1. الصف الأولي: []

2. الصف بعد إضافة x و y: [4, 5]

3. قراءة العنصر الأمامي: x = 4

- الصف بعد القراءة: [5]

4. الصف بعد إضافة x + 5 و 16 و x و y - 3: [5, 9, 16, 4, 2]

5. طباعة عناصر الصف:

- Queue Elements:

- طباعة العنصر الأمامي 5

- الصف بعد الإزالة: [9, 16, 4, 2]

- طباعة العنصر الأمامي 9

- الصف بعد الإزالة: [16, 4, 2]

- طباعة العنصر الأمامي 16

- الصف بعد الإزالة: [4, 2]

- طباعة العنصر الأمامي 4

- الصف بعد الإزالة: [2]

- طباعة العنصر الأمامي 2

- الصف بعد الإزالة: []

بالتالي، ستكون النتيجة النهائية للشفرة هي:

Queue Elements:

5

9

16

4

2

Q2\

أفترض أن الشفرة تهدف إلى استخدام صف مرتبط (linked queue) لتخزين الأعداد.

لنتابع الشفرة خطوة بخطوة:

1. تعريف صف الأعداد المرتبط وإضافة العناصر إليه:

- تعريف صف الأعداد المرتبط queue باستخدام `linkedQueue` دون تحديد النوع، لذا سنفترض أنها `linkedQueue<int>`.

- إضافة العدد 10 إلى الصف باستخدام `queue.enqueue(10)`.

- إضافة العدد 20 إلى الصف باستخدام `queue.enqueue(20)`.

2. قراءة العنصر الأمامي وإزالته من الصف:

- قراءة العنصر الأمامي من الصف وطباعته باستخدام `cout << queue.front() << endl`.

- إزالة العنصر الأمامي من الصف باستخدام `queue.dequeue()`.

3. إضافة العناصر إلى الصف:

- إضافة العدد (2 \* العنصر الأخير في الصف) إلى الصف باستخدام `queue.enqueue(2 \* queue.back())`.

- إضافة العنصر الأمامي إلى الصف باستخدام `queue.enqueue(queue.front())`.

- إضافة العدد 5 إلى الصف باستخدام `queue.enqueue(5)`.

- إضافة (العنصر الأخير في الصف - 2) إلى الصف باستخدام `queue.enqueue(queue.back() - 2)`.

4. إنشاء صف مؤقت وتعيينه إلى قيمة الصف الأصلي:

- إنشاء صف مؤقت tempQueue باستخدام `linkedQueue<int> tempQueue`.

- تعيين tempQueue ليكون نسخة من قيمة الصف الأصلي باستخدام `tempQueue = queue`.

5. طباعة العناصر في tempQueue وإزالتها:

- بدء حلقة while لطباعة العناصر المتبقية في tempQueue:

- التحقق مما إذا كان tempQueue فارغًا أم لا باستخدام `!tempQueue.isEmptyQueue()`.

- طباعة العنصر الأمامي لـ tempQueue باستخدام `System.out.println(tempQueue.front())`.

- إزالة العنصر الأمامي من tempQueue باستخدام `tempQueue.dequeue()`.

6. طباعة العنصر الأمامي والعنصر الأخير في الصف الأصلي:

- طباعة العنصر الأمامي في الصف الأصلي باستخدام `System.out.println(queue.front())`.

- طباعة العنصر الأخير في الصف الأصلي باستخدام `System.out.println(queue.back())`.

الآن دعونا نحلل الشفرة خطوة بخطوة:

1. الصف الأولي: [10, 20]

2. قراءة العنصر الأمامي: العنصر الأمامي = 10

- الصف بعد الإزالة: [20]

- الناتج المطبوع: 10

3. الصف بعد إضافة (2 \* العنصر الأخير) والعنصر الأمامي و 5 و (العنMy apologies! It seems that there was a formatting issue in your question. The code segment you provided is not valid Java syntax. Could you please double-check and provide the correct code segment so that I can assist you further?

Q3\

الشفرة تبدو كتلاعب بكومبينيشن من الهياكل البيانية (stack و queue) لمعالجة سلسلة من الأعداد المدخلة.

لنتابع الشفرة خطوة بخطوة:

1. تعريف الهياكل البيانية:

- تعريف الستاك stack باستخدام `ArrayStack<int> stack`.

- تعريف الكيو queue باستخدام `ArrayQueue<int> queue`.

- تعريف المتغير x بدون تهيئة، لذا قيمته تعتمد على القيمة الافتراضية لنوع int.

2. قراءة الأعداد ومعالجتها:

- إضافة العدد 0 إلى الستاك باستخدام `stack.push(0)`.

- إضافة العدد 0 إلى الكيو باستخدام `queue.enqueue(0)`.

- طباعة قيمة المتغير x باستخدام `System.out.println(x)`. قيمة x حاليًا غير معروفة، لذا سيتم طباعة القيمة الافتراضية لنوع int وهي 0.

3. حلقة while لمعالجة الأعداد:

- التحقق مما إذا كانت قيمة x ليست -999 باستخدام `while (x != -999)`.

- استخدام عملية القسمة المتبقية لـ x على 4 باستخدام `x % 4`.

- حالة 0: إضافة x إلى الستاك باستخدام `stack.push(x)`.

- حالة 1: التحقق مما إذا كان الستاك غير فارغ باستخدام `!stack.isEmptyStack()`.

- إذا كان الستاك غير فارغ، طباعة "Stack Element =" وقيمة العنصر العلوي للستاك باستخدام `System.out.println("Stack Element =")` و `System.out.println(stack.top())`، ثم إزالة العنصر العلوي من الستاك باستخدام `stack.pop()`، وإلا طباعة "Sorry, the stack is empty."

- حالة 2: إضافة x إلى الكيو باستخدام `queue.enqueue(x)`.

- حالة 3: التحقق مما إذا كان الكيو غير فارغ باستخدام `!queue.isEmptyQueue()`.

- إذا كان الكيو غير فارغ، طباعة "Queue Element =" وقيمة العنصر الأمامي للكيو باستخدام `System.out.println("Queue Element =")` و `System.out.println(queue.front())`، ثم إزالة العنصر الأمامي من الكيو باستخدام `queue.dequeue()`، وإلا طباعة "Sorry, the queue is empty."

- طباعة قيمة المتغير x بعد المعالجة.

4. طباعة عناصر الستاك:

- طباعة "Stack Elements: ".

- حلقة while لطباعة العناصر المتبقية في الستاك:

- التحقق مما إذا كان الستاك غير فارغ باستخدام `!stack.isEmptyStack()`.

- طباعة العنصر العلوي للستاك باستخدام `System.out.println(stack.top())`.

- إزالة العBased on the provided code, it seems to be incomplete and contains several errors. Here is a modified version of the code that should work correctly:

```java

ArrayStack<Integer> stack = new ArrayStack<>();

ArrayQueue<Integer> queue = new ArrayQueue<>();

int x;

stack.push(0);

queue.enqueue(0);

System.out.println(x);

while (x != -999) {

switch (x % 4) {

case 0:

stack.push(x);

break;

case 1:

if (!stack.isEmptyStack()) {

System.out.println("Stack Element = " + stack.top());

stack.pop();

} else {

System.out.println("Sorry, the stack is empty.");

}

break;

case 2:

queue.enqueue(x);

break;

case 3:

if (!queue.isEmptyQueue()) {

System.out.println("Queue Element = " + queue.front());

queue.dequeue();

} else {

System.out.println("Sorry, the queue is empty.");

}

break;

}

System.out.println(x);

}

System.out.println("Stack Elements: ");

while (!stack.isEmptyStack()) {

System.out.println(stack.top());

stack.pop();

}

System.out.println("Queue Elements: ");

while (!queue.isEmptyQueue()) {

System.out.println(queue.front());

queue.dequeue();

}

```

In this code, an `ArrayStack` and an `ArrayQueue` are initialized, and the variable `x` is declared. The code then pushes 0 to the stack, enqueues 0 to the queue, and prints the value of `x` (which will be the default value of 0).

The while loop continues until `x` is -999. Within the loop, the value of `x` is checked modulo 4, and based on the result, different actions are taken:

- Case 0: `x` is pushed to the stack.

- Case 1: If the stack is not empty, the top element is printed and removed from the stack. Otherwise, a message indicating an empty stack is printed.

- Case 2: `x` is enqueued to the queue.

- Case 3: If the queue is not empty, the front element is printed and dequeued. Otherwise, a message indicating an empty queue is printed.

After each iteration of the loop, the value of `x` is printed.

Finally, the remaining elements in the stack and queue are printed and removed using the respective while loops.

Please note that the code assumes the existence and correct implementation of the `ArrayStack` and `ArrayQueue` classes.

Q4\

أ) بعد إضافة عنصر إلى الطابور (queue)، ستتم زيادة قيمة `queueRear` بمقدار واحد. إذا وصلت قيمة `queueRear` إلى نهاية المصفوفة (في هذه الحالة الفهرس 99)، فإنها ستعود إلى بداية المصفوفة. لذا، قيمة `queueRear` الجديدة ستكون 0. قيمة `queueFront` ستبقى ثابتة عند 50.

لذا، بعد إضافة عنصر إلى الطابور، القيم ستكون:

- `queueFront`: 50

- `queueRear`: 0

ب) بعد إزالة عنصر من الطابور، ستتم زيادة قيمة `queueFront` بمقدار واحد. إذا وصلت قيمة `queueFront` إلى نهاية المصفوفة، فإنها ستعود أيضًا إلى بداية المصفوفة. لذا، قيمة `queueFront` الجديدة ستكون 51. قيمة `queueRear` ستبقى ثابتة عند 99.

لذا، بعد إزالة عنصر من الطابور، القيم ستكون:

- `queueFront`: 51

- `queueRear`: 99

Q5\

أ) بعد إضافة عنصر إلى الطابور (queue)، ستتم زيادة قيمة queueRear بمقدار واحد. إذا وصلت قيمة queueRear إلى نهاية المصفوفة (في هذه الحالة الفهرس 99)، فإنها ستعود إلى بداية المصفوفة. لذا، قيمة queueRear الجديدة ستكون 26. قيمة queueFront ستبقى ثابتة عند 99.

لذا، بعد إضافة عنصر إلى الطابور، القيم ستكون:

queueFront: 99

queueRear: 26

Q6\

أ) بعد إضافة عنصر إلى الطابور (queue)، ستتم زيادة قيمة `queueRear` بمقدار واحد. إذا وصلت قيمة `queueRear` إلى نهاية المصفوفة (في هذه الحالة الفهرس 99)، فإنها ستعود إلى بداية المصفوفة. لذا، قيمة `queueRear` الجديدة ستكون 76. قيمة `queueFront` ستبقى ثابتة عند 25.

لذا، بعد إضافة عنصر إلى الطابور، القيم ستكون:

- `queueFront`: 25

- `queueRear`: 76

ب) بعد إزالة عنصر من الطابور، ستتم زيادة قيمة `queueFront` بمقدار واحد. إذا وصلت قيمة `queueFront` إلى نهاية المصفوفة، فإنها ستعود أيضًا إلى بداية المصفوفة. لذا، قيمة `queueFront` الجديدة ستكون 26. قيمة `queueRear` ستبقى ثابتة عند 75.

لذا، بعد إزالة عنصر من الطابور، القيم ستكون:

- `queueFront`: 26

- `queueRear`: 75

Q7\

أ) بعد إضافة عنصر إلى الطابور (queue)، ستتم زيادة قيمة `queueRear` بمقدار واحد. إذا وصلت قيمة `queueRear` إلى نهاية المصفوفة (في هذه الحالة الفهرس 99)، فإنها ستعود إلى بداية المصفوفة. ومن المعطى أن قيمة `queueRear` هي 99، وهي الفهرس الأخير في المصفوفة.

بما أن المصفوفة مكتملة وليس هناك مساحة فارغة لإضافة المزيد من العناصر، فإضافة عنصر جديد إلى الطابور سيؤدي إلى تجاوز الحجم المسموح به للمصفوفة وبالتالي حدوث خطأ.

لذا، ستظل قيم `queueFront` و `queueRear` دون تغيير:

- `queueFront`: 99

- `queueRear`: 99

ب) بعد إزالة عنصر من الطابور، ستتم زيادة قيمة `queueFront` بمقدار واحد. إذا وصلت قيمة `queueFront` إلى نهاية المصفوفة، فإنها ستعود إلى بداية المصفوفة. ومن المعطى أن قيمة `queueFront` هي 99.

بما أن الطابور فارغ ولا توجد عناصر لإزالتها، فإزالة عنصر من الطابور سيؤدي إلى حدوث خطأ.

لذا، ستظل قيم `queueFront` و `queueRear` دون تغيير:

- `queueFront`: 99

- `queueRear`: 99

Q8\

إليك نموذجًا لتنفيذ الدالة `reverseQueue` بلغة Python:

```python

def reverseQueue(queue):

stack = [] # إنشاء مكدس فارغ

# دفع جميع العناصر من الطابور إلى المكدس

while not queue.isEmpty():

stack.append(queue.dequeue())

# سحب العناصر من المكدس وإعادة إضافتها إلى الطابور

while len(stack) > 0:

queue.enqueue(stack.pop())

# تم عكس العناصر في الطابور

```

في هذا التنفيذ، نستخدم مكدس لعكس عناصر الطابور. أولاً، نقوم بإخراج جميع العناصر من الطابور ونقوم بدفعها إلى المكدس. ثم، نقوم بسحب العناصر من المكدس وإعادة إضافتها إلى الطابور. بعد هذه العملية، ستكون العناصر في الطابور قد تم عكسها.

يرجى ملاحظة أن دوال `isEmpty()`، `dequeue()` و `enqueue()` قد تختلف اعتمادًا على تنفيذ كائن الطابور المستخدم. تأكد من ضبط استدعاءات تلك الدوال وفقًا لتنفيذ الطابور الذي تستخدمه.

Q9\

لحساب الحجم الحالي للطابور Q، نحتاج إلى مراعاة عدد عمليات الإضافة (enqueue) والإزالة (dequeue) التي تم تنفيذها على الطابور.

إجمالاً، تم تنفيذ 32 عملية إضافة عناصر إلى الطابور.

تم تنفيذ 15 عملية إزالة من الطابور، ومن بينها 5 أعادت قيمة فارغة (null) للإشارة إلى أن الطابور فارغ. هذا يعني أنه تمت إزالة 15 - 5 = 10 عناصر بنجاح من الطابور.

لحساب الحجم الحالي للطابور، نقوم بطرح عدد العناصر التي تمت إزالتها من عدد العناصر التي تمت إضافتها:

الحجم الحالي للطابور Q = عدد العناصر المضافة - عدد العناصر المزالة

= 32 - 10

= 22

بالتالي، الحجم الحالي للطابور Q هو 22.

Q10\

تتمثل الخطوات المطلوبة على النحو التالي:

1. يتم إنشاء deque فارغًا.

2. addFirst(3): [3]

3. addLast(8): [3, 8]

4. addLast(9): [3, 8, 9]

5. addFirst(1): [1, 3, 8, 9]

6. last(): 9

7. isEmpty(): False

8. addFirst(2): [2, 1, 3, 8, 9]

9. removeLast(): 9 (تمت إزالة العنصر الأخير، وهو 9)

10. addLast(7): [2, 1, 3, 8, 7]

11. first(): 2

12. last(): 7

13. addLast(4): [2, 1, 3, 8, 7, 4]

14. size(): 6

15. removeFirst(): 2 (تمت إزالة العنصر الأول، وهو 2)

16. removeFirst(): 1 (تمت إزالة العنصر الأول، وهو 1)

بالتالي، القيم المُرَجَّعَة خلال هذه السلسلة من عمليات deque هي كالتالي:

[3]

[3, 8]

[3, 8, 9]

[1, 3, 8, 9]

9

False

[2, 1, 3, 8, 9]

9

[2, 1, 3, 8, 7]

2

7

[2, 1, 3, 8, 7, 4]

6

2

1