Helwan University
Faculty of Computers and Information

Midterm
Closed Book
1 hours

Department of Computer Science
Data Structures
Fall 2012

MCQ: (1 mark on each).
4 wrong answers cancel one correct answer.

1. Consider the linked implementation of queues that was studied in lectures. the function `B` in the following code:

```
void B(Queue *pq){
    pq->front=NULL;
    pq->rear=NULL;
}
void A(Queue *pq){
    pq->rear=pq->front;
    while(pq->rear){
        pq->rear=pq->rear->next;
        free(pq->front);
        pq->front=pq->rear;
    }
}
```

a) is faster than the function `A`, but it leaves the memory occupied with data that is neither needed nor reachable any more.
b) has a syntax error.
c) all of the above.
d) exactly does as what function `A` does.
e) non of the above.

2. 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;`
e) `ps->entry[ps->top--]=e;`

3. In array-based implementation of circular queues, the ratio between the time needed for executing the function `QueueSize`, while the queue has 10 elements, to the time needed for executing the same function while the queue has only 5 elements is:

a) It depends on the element type.
b) It depends on the number of elements.
c) It depends on `QUEUEMAX` (the size of the array.)
d) 2.
e) none of the above.

```
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.

4. 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;`
e) none of the above.

5. 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;`
e) `none of the above;`

6. Assume that we define `typedef StackNode *Stack` in the implementation level, and declare `Stack s` in the user level. Then, if the statement `if (s) s->entry=e;` is written in the user level it

a) causes a syntax error, since it accesses the details of the implementation from the user level.
b) causes a runtime error after the program starts.
c) causes a problem, specially if the stack is not empty.
d) all of the above.
e) violates the concept of ADT (Abstract Data Type), i.e., hiding the details; but will execute without problems.

```
CreateStack(&tmps);
CreateStack(&s1);
CreateQueue(&q1);
while(!StackEmpty(&s)){
    Pop(&e, &s);
    Push(e, &s1);
    Push(e, &tmps);
}
while(!StackEmpty(&tmps)){
    Pop(&e, &tmps);
    Append(e, &q1);
}
```

Code 2.

7. Assume that we executed the code of Code 2. Then, whether we `Pop` from `s1` or `Serve` from `q1`, we will retrieve the elements

- **a)** in a similar order, which is the same order that we would get by popping from `s`.
- **b)** in a similar order but opposite to the order that we would get by popping from `s`.
- **c)** in a reverse order.
- **d)** not in a particular order.
- **e)** none of the above, since the elements in `s1` will be different from those in `q1`.

```
int Fib(int n){
    if (n<=0) return 0;
    if (n==1) return 1;
    return Fib(n-1)+Fib(n-2);
}
```

Code 3.

8. The recursive function in Code 3 returns the Fibonacci number of an integer n. Calculating the Fibonacci of 4 results in calling the function (including the first call `Fib(4)`)

- **a)** 2 times.
- **b)** 3 times.
- **c)** 5 times.
- **d)** 8 times.
- **e)** 9 times.

9. For a linked queue, the function

```
void A(Queue *pq, QueueEntry *pe){
    *pe=pq->front->entry;
}
```

- **a)** gets a copy from the first element.
- **b)** is $\Theta(1)$;
- **c)** all of the above.
- **d)** is the `Dequeue` function.
- **e)** should not accept the pointer `pq` because the queue is not changed inside. Rather, it should accept the queue itself, i.e., `void A(Queue q, ···)`.