Midterm Examination Problem Sheet

TIME: 04/24/2012, 14:20–17:20

This is a open-book exam. You can use any printed materials as your reference during the exam. Any electronic devices are not allowed.
Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
Both English and Chinese (if suited) are allowed for answering the questions. We do not accept any other languages.

There are 10 questions in the exam, each worth 20 points—the full credit is 200 points. For the 10 questions, 3 of them are marked with * and are supposedly simple; 4 of them are marked with ** and are supposedly regular; 3 of them are marked with *** and are supposedly difficult. The questions are roughly ordered by the difficulty level.

(1) (20%, *) George and Mary decide to be together forever (a.k.a. 一直走下去). To symbolize their love to each other, they decide to exchange their very first gifts. For the following code, the C++ function `exchange` is supposed to do the task, but the result appears incorrect.

(a) Illustrate (with drawing) what the memory layout is for `George`, `Mary`, `boy`, `girl` at `//DRAW`.

(b) Modify the code to complete the exchange successfully, **without** using any pointers. Please simply highlight which line you want to change (you only need to change one!), and write down how you want to change it.

```
1  typedef GIFT int;
2  struct Person{ GIFT gift; };
3
4  void exchange(Person boy, Person girl){
5    GIFT onTable = girl.gift;
6    girl.gift = boy.gift;
7    boy.gift = onTable;
8    //DRAW
9  }
10
11 Person George, Mary;
12
13 int main(){
14   //some more code to let George and Mary prepare gifts
15   exchange(George, Mary);
16   //now George should take Mary's gift, and Mary takes George's
17   return 0;
18 }
```

(2) (20%, *) Considering representing a queue by a fixed-size circular array of size 4, where the front and the tail of the (empty) queue are both at 0 in the beginning. After executing each step of the following operations sequentially, list the contents of the array, as well as where head and tail are. For locations without contents, please use a special character to represent them.

```
enqueue(1); enqueue(3); dequeue(); enqueue(2); enqueue(4);
 dequeue(); enqueue(5); enqueue(6); dequeue(); dequeue();
```

(3) (20%, **) Consider having two stacks for storing integers, a `red` one and a `blue` one with the `push` and `pop` operations only. Illustrate how to implement the `int pop_mth(color s, int m)` operation that pops the m-th element from the top (and only that element) from stack s. Note that `pop_mth(s, 1)` should be equivalent to the usual pop from stack s, and after the operation, all other elements should remain in their original orders.

(4) (20%, **) The following code causes illegal memory access.

(a) Illustrate (with drawing) what the memory layout is at `//DRAW`.

(b) Explain why the `//SEGFAULT` line could lead to illegal memory access.

```
class VectorBad{
public:
  int* data;
  VectorBad(){ data = new int[2]; }
  ~VectorBad(){ delete[] data; }
  }
};

int main(){
  VectorBad a; a.data[0] = 1; a.data[1] = 3;
  {
    VectorBad temp = a;
    for(int i=0;i<2;i++) temp.data[i] += (i * i);
    //DRAW
  }
  a.data[1] = 5; //SEGFAULT
  return 0;
}
```

(5) (20%, **) The following recursive code reverses a singly-linked list.

```
void reverseList(Node* head){
  if (head->next){
    Node* tail = head->next;
    reverseList(head->next)
    tail->next = head;
  }
}
```

(a) Illustrate how the code reverses a linked list `A->B->C` when calling `reverseList(A)`. You need to describe the steps clearly for the TAs.

(b) Complete the following tail-recursive code that also reverses the linked list.

```
void reverseListAndAppend(Node* head, Node* append){
  if (head->next){
    Node* tmp = head->next;
    head->next = append;
    //what to put here? reverseListAndAppend(..., ...);
  }
  else{
    //what to put here?
  }
}
void reverseList(Node* head){
  //what to put here? reverseListAndAppend(..., ...)
}
```

(6) (20%, *) The lower-triangular matrix is a matrix M with $M[r][c] = 0$ whenever $c > r$. Naturally, we do not want to waste storage on the 0 part. A common way of storing a lower-triangular matrix is called a "rectangular representation." For instance, the content of a 4 by 4 lower triangular matrix $\begin{matrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 4 & 5 & \mathbf{6} & 0 \\ 7 & 8 & \mathbf{9} & \mathbf{10} \end{matrix}$ can fit in a 5 by 2 rectangular matrix $\begin{matrix} \mathbf{10} & \mathbf{9} \\ 1 & \mathbf{6} \\ 2 & 3 \\ 4 & 5 \\ 7 & 8 \end{matrix}$.

Following the example above, design a data structure that implements the rectangular storage scheme for an arbitrary lower-triangular matrix (*Hint: discuss cases of even or odd number of columns*). Illustrate the scheme clearly and discuss how to access the element at row r and column c of the lower-triangular matrix.

For the following two questions, you can only use this definition:
Let f, g be functions fron nonnegative integers to real numbers. We say $f(n) = O(g(n))$ if there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for $n \geq n_0$.

(7) (20%, **) Prove or disapprove the following statement. "For non-negative functions f, g, h, if $f(n) = O(g(n))$, then $f(n) + h(n) = O(g(n) + h(n))$."

(8) (20%, ***) Prove or disapprove the following statements.

(a) "If $a < 1$ or ($a = 1$ and $b \leq 0$), then $2^{an^2+bn+c} = O(2^{n^2})$."

(b) "If $2^{an^2+bn+c} = O(2^{n^2})$, then $a < 1$ or ($a = 1$ and $b \leq 0$)."

(9) (20%, ***) A linked list allows a node to link to the next one, which can be slow if you want to access a node far far away. Consider the following multiply-linked list, which allows a node store K pointers, each linking to next-1, next-2 (a node two steps away), next-4 (a node four steps away), next-8, $\cdots$, next-2^{K-1}. Finish the code for `getDataNext`. Which walks s steps from the current node and then return the pointer to the destination node, and $NULL$ otherwise. Your code should run within $O(\log N)$ of time complexity, where N is the number of elements in the multiply-linked list. Briefly explain why your code runs within $O(\log N)$.

```
#define K (32)
class MLNode{
public:
  int data;
  MLNode next[K]; //next[i] links to a node (2^i) steps away
                  //or NULL if the node does not exist

  MLNode* getDataNext(unsigned int s){
    if (s == 0) return this;
    //FINISH THE REST OF THE CODE
  }
};
```

(10) (20%, ***) Consider an array a of N integers $a_0 < a_1 < a_2 < \cdots < a_M$ and $a_M > a_{M+1} \cdots > a_{N-1}$. In other words, a_M is the maximum element of the array. Write down an $O(\log N)$-time algorithm that finds the value of a_M. Briefly describe why the algorithm is correct. (*Hint: think about binary search.*)