# Dept. of Computer Science and Engineering, undergraduate
# National Sun Yat-sen University
# Data Structures - Middle Exam, Nov. 28, 2011

1. Assume that each *int* element of an array $a$ occupies 4 units of storage and each $float$ element of an array $a$ occupies 8 units of storage. Suppose that the first element of $a$ is $a[0][0]$ or $a[0][0][0]$ and its address is 200. Please give the address of the indicated element in each of the following cases. (10%)

   (a) int a[7][10]; row-major order; find element $a[4][6]$.

   (b) float a[7][10]; column-major order; find element $a[4][6]$.

   (c) float a[5][6][4]; row-major order; find element $a[3][4][2]$.

   (d) int a[5][6][4]; column-major order; find element $a[3][4][2]$.

2. Suppose that a matrix $m[\,][\,]$ is stored in a linear array $a[\,]$ with the following sequence:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 5 | 6 | 14 | $\cdots$ |
| 2 | 4 | 7 | 13 | $\cdots$ | |
| 3 | 8 | 12 | $\cdots$ | | |
| 9 | 11 | $\cdots$ | | | |
| 10 | $\cdots$ | | | | |

   Please give the mapping function from $m[i][j]$ to $a[k]$. Note that the upper left corner of $m$ is the first element $m[0][0]$, the first element of $a$ is $a[0]$, $m[0][1] = 1$ and $m[0][2] = 5$. (8%)

3. How do you implement the *set* operations (including *intersection, union, difference* and *containment*) in a programming language? Give examples to illustrate your implementation. (10%)

4. (a) Transform the *prefix* expression $++A-*-BCDE/+FG-HI$ to *infix* and *postfix* expressions. Draw its expression tree. (9%)

   (b) Give a rule to determine whether an expression is a postfix expression or not. (6%)

5. The three operations *push, pop,* and *no-op* (i.e. forward the input to the output directly) can be used to rearrange an input sequence. For example, a sequence of operations *push, push, no-op, pop, pop* will rearrange 123 to 321. For six numbers 1,2,3,4,5,6 entered in that order, which of the following rearrangements are (is) not possible? (A) 325641 (B) 132546 (C) 154623 (D) 235146 (E) 324156. (8%)

6. Suppose that we have a language defined as follows:

   $expression \rightarrow term + term | term$

   $term \rightarrow factor * factor | factor$

   $factor \rightarrow letter | (expression)$

   $letter \rightarrow A|B|C|D$

   where the vertical bar | means "or". Answer YES or NO for the following questions. (9%)

(a) Is $A+B$ an *expression*? a *term*? a *factor*?

(b) Is $A*(B)$ an *expression*? a *term*? a *factor*?

(c) Is $A+B+C$ an *expression*? a *term*? a *factor*?

7. What are the advantage and disadvantage of a linked list implemented by an array and dynamic variables, respectively? (6%)

8. Write a *recursive* C function to compute the average of the elements of an array. Note that you should write only one function and the length of the array is assumed to be unknown. The upper bound of the length is $N$, but the length is not necessarily equal to $N$. (10%)

```
#define N 100
int a[N]; /* the array storing the elements */
int avg(···)
```

9. Write a C function to insert a new element (integer) with value $x$ into a sorted list, which is stored in a linearly linked list with implementation of an array. Note that the smallest element is stored in the front node of the linked list. It is assumed that the list before the insertion contains at least one element. (12%)

```
struct nodetype {
    int info;
    int next;
}
struct nodetype node[100];
void place(int *list, int x)
```

You can call the following two functions directly. In other words, you need not write the programs of these two functions.
(1) *void push(list, x)*: insert a node with value $x$ into the front of the list.
(2) *void insafter(q, x)*: insert a node with value $x$ after node $q$ of the list.

10. Write a C function to combine two ordered lists into a single ordered list, where these lists are represented with circular doubly linked lists and implemented by dynamic variables. Note that each list may be empty. (12%)

```
struct nodetype {
    int info;
    struct nodetype *left, *right;
}
typedef struct nodetype *NODEPTR;
void combine(NODEPTR *lista, NODEPTR *listb)
/* *lista and *listb are the pointers of the two lists */
/* After combination, *lista points to the resulting list. */
/* The external pointer points to the tail, not the head, in a circular linked list. */
```