Department of Computer Science and Engineering
National Sun Yat-sen University
Data Structures - Middle Exam, Nov. 20, 2017

1. Suppose an array is declared as *a*[5][6][4], where the address of *a*[0][0][0] is 200 and each element requires four bytes. Please give the addresses of *a*[3][4][2] with the row-major representation and the column-major representation. (8%).

2. What are printed by each of the following C programs? (16%)

   (a)
   ```
   char e=13;
   printf("%d \n",~((e+4) >> 3));
   ```
   (b)
   ```
   void f(int a[ ], int b[ ], int *c, int *d)
   {    printf("%d %d %d %d \n", a[1],b[2],*(c+2),d[4]);    }
   int main( )
   {    int e[ ]={20,21,22,23,24,25,26,27,28,29,30};
        f(e,e+3,&e[2],&e[1]+4); }
   ```
   (c)
   ```
   int a[ ]={11,14,17,20,23,26};    int *p;
   p=a;    *(p++)=5;    (*(++p))++;
   printf("%d %d %d %d \n",a[0],a[1],*p,*(p+2));
   ```
   (d)
   ```
   union {
       char m;
       unsigned char n;
   }u;
   u.n=193;
   printf("%d \n",u.m);
   ```

3. Please draw the expression tree of the infix expression (A+B)*D+E/(F+A*D)+C, and then give the prefix and postfix forms. (9%)

4. John is learning numeric symbols (1,2,3,4…), but sometimes he may write 1 as L. With only the first three digits (1, 2, 3, L) for addition, we want to know the number of permutations whose sum is *n*. For example, if *n*=2, the 5 permutations have the same sum 2: 11, 1L, L1, LL, 2. If *n*=3, the 13 permutations have the same sum 3: 111,11L, 1L1,1LL, L11, L1L, LL1, LLL, 21,2L, 12, L2,3. Let $f(n)$ denote the number of permutations with sum *n*. Then $f(n)$ can be calculated by the recurrence formula: $f(n) = a\times f(n\text{-}1)+b\times f(n\text{-}2)+c\times f(n\text{-}3)$, $n\geq 4$. What are the values of *a*, *b* and *c*? (9%)

5. Suppose that a matrix *m*[ ][ ] is stored in a linear array *a*[ ] with the sequence in the following figure. Please give the mapping function from *m*[*i*][*j*] to *a*[*k*], that is, to express *k* as a function of *i* and *j*. Note that the upper left corner of *m* is the first element *m*[0][0], the first element of *a* is *a*[0]. And *m*[0][1] = 1, *m*[0][2] = 5, and … (10%)

0	1	5	6	14	…
2	4	7	13		
3	8	12	…		
9	11	…			
10	…				

6. Explain each of the following terms. (12%)
   (a) $O(n^2)$
   (b) protected in C++ language
   (c) sparse matrix

7. Write a recursive C/C++ function to perform the *binary search* on a nondecreasingly sorted array. (12%)

```
int BSearch(int a[ ], int x, int left, int right)
// a[ ]: nondecreasingly sorted array
// search for x in a[left], a[left+1], …, a[right-1], a[right]
//Return the index if found. Return -1 if not found.
{
```

Please write the body of BSearch( ).

```
} // end of BSearch( )
```

8. Write a C/C++ function to perform *insert* (into the rear) and *remove* (from the front) operations of a circular queue implemented with an array. (12%)

```
int front, rear;            // front, rear pointers
int capacity=100;           // size of queue
char q[100];                // array for the circular queue.
            //No data element is stored in a[front], but a[rear] stores one element.
void Insert(char x)
     // insert x into the rear. You have to check if q is full before the insertion.
{
```

(a) Please write the body of Insert( ).

```
} // end of insert ( )
char Remove(void)
// Remove an element from the front, and return the removed element.
// You have to check if q is empty before the removal.
{
```

(b) Please write the body of Remove( ).

```
} // end of Remove( )
```

9. Let x=(x_1, x_2, …, $x_{m\text{-}1}$, x_m) and y=(y_1, y_2, …, $y_{n\text{-}1}$, y_n) be two circular chains. Write a C++ function to concatenate the two circular chains into a circular chain z=(x_1, x_2, …, $x_{m\text{-}1}$, x_m, y_1, y_2, …, $y_{n\text{-}1}$, y_n). Note that x or y may be empty. (12%)

```
class ChainNode {
   int data;
   ChainNode *link;    // Point to the next node
};
class Chain {
   ChainNode *first *last;      // circular chain
   Chain concatenate(Chain &y )
        // y is concatenated to the end of *this (x)
        // You have to consider empty chains.
   {
   Chain z;          // The resulting chain
```

Please write the body of concatenate( ).

```
   return z;
   } // end of concatenate( )
};
```

Answer:

1.

row-major:

200 + 4 * (3 * 6 * 4 + 4 * 4 + 2)
= 200 + 4 * 90 = 560

column-major:

200+ 4* (2 * 6 * 5 + 4 * 5 + 3)
= 200 + 4 * 83 = 532

2.

(a) ~((13 + 4) >> 3) = ~(17 >> 3) = ~ (($00010001)_2$ >> 3)
= ~(($00000010)_2$) = ($11111101)_2$ = -3

由於 e 是 char 資料型態，故以 8 bits 呈現。印出 e 時，是以%d 表現，亦即是帶有正負號之整數，因此需將當時的數值解讀為 2's complement。

Output: -3

(b) a[0]對應至 e[0]，因此 a[1] = e[1] = 21
b[0]對應至 e[3]，因此 b[2] = e[5] = 25
c 對應至 e[2](也就是 c[0]對應至 e[2])，因此*(c+2) = e[2+2] = 24
d 對應至 e[5] (也就是 d[0]對應至 e[5])，因此 d[4] = e[9] =29

Output: 21 25 24 29

(c) p=a; // p 對應至 a[0]
*(p++) = 5; //先執行 *p =5，因此 a[0]=5。再做 p=p+1，因此 p 對應至 a[1]
(*(++p))++; //先做 p=p+1，因此 p 對應至 a[2]。再做(*p)++，即 a[2]=17+1=18
a[0] = 5
a[1] = 14
*p =a[2]=18
*(p+2) a[4]= 23

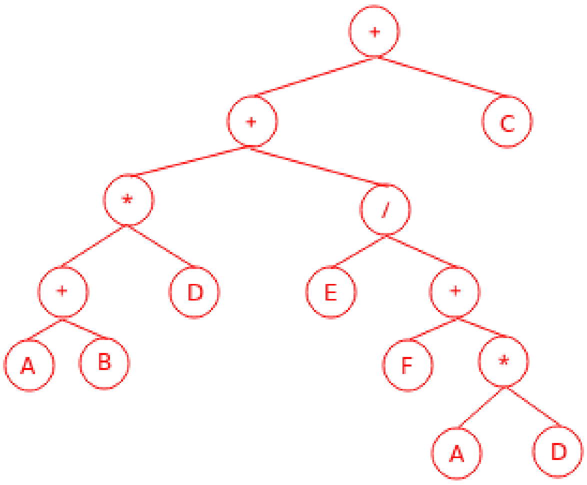
Output: 5 14 18 23

(d) u 與 v 佔據相同記憶體位置，兩者的資料內容相同，但解讀方式不同。
u 為無正負號的 char，亦即為 8 bit 無正負號之整數。
m 亦是 8 bit，但帶有正負號。
將 193 轉換成 2's complement 之負值即可。
u=$193_{(10)}$ = $11000001_{(2)}$, 2's complement 為 $63_{(10)}$ = $00111111_{(2)}$ ，
故 $11000001_{(2)}$ 為-63

Output: -63

Summary: (a) -3 (b) 21 25 24 29 (c) 5 14 18 23 (d) -63

3. Prefix: ++*+ABD/E+F*ADC
Postfix: AB+D*EFAD*+/+C



4. f(n) 可以表示如下：

f(n) = 2f(n-1) + f(n-2) + f(n-3)

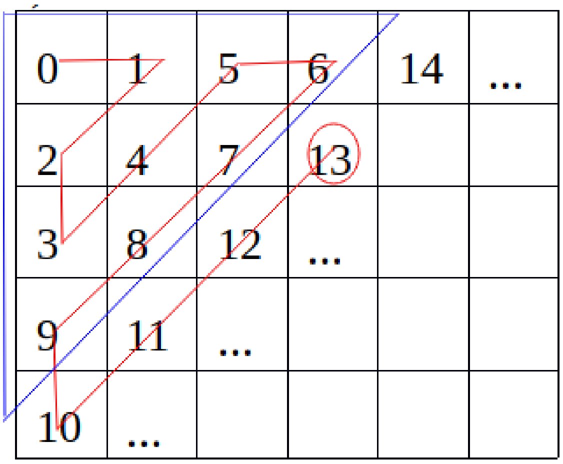
2f(n-1) 在 f(n-1)每項之後再加上 1 及 L，可使總和由 n-1 增加為 n

f(n-2) 在 f(n-2)每項之後再加上 2，可使總和由 n-2 增加為 n

f(n-3) 在 f(n-3)每項之後再加上 3，可使總和由 n-3 增加為 n

Summary: a=2, b=1, c=1

5.



0	1	5	6	14	…
2	4	7	13		
3	8	12	…		
9	11	…			
10	…				

計算[i][j]的編號時，在此之前的斜線（圖中藍色三角形），共有如下的數字個數：

1+2+3+…+(i+j)= (i+j)(i+j+1) / 2

之後再依 i + j 為奇數或偶數偶，判斷[i][j]所在斜線，尚需增加之個數（例如圖中[1][3]位置，尚需增加 j=3，因為 i+j 為偶數)。完整公式如下：

k=[1+2+…+(i+j)]+i= (i+j)(i+j+1) / 2 +i，if i+j is odd (奇數)

k=[1+2+…+(i+j)]+j= (i+j)(i+j+1) / 2 +j，if i+j is even (偶數)

6.

(a) O(n^2)：至多與 n^2 成正比，可用來表示時間複雜度或空間複雜度。

(b) 能被原本的 class 以及衍生的 class(繼承者)存取。

(c) 一個矩陣中，大部分元素為零，少數為非零。

7.

```
if(left > right)
      return -1;
int mid = (left + right)/2;
if(a[mid] == x)
      return mid;
if(a[mid] > x)
      Bsearch(a, x, left, mid-1);
if(a[mid] < x)
      Bsearch(a, x, mid +1, right);
```

8.

```
(a)
      if((rear + 1)%capacity) == front)
            throw “full”;
      rear = (rear + 1)%capacity;
      q[rear] = x;
(b)
      if(rear == front)
            throw “empty”;
      front = (front + 1)%capacity;
      return q[front];
```

9.

```
if( front == NULL ){    // x is NULL
      z.first = y.first;
      z.last = y.last;
}
else if( y.first == NULL ){ // y is NULL
      z.first = first;
      z.last = last;
}
else{    // Both x and y are not NULL
      last→link = y.first;    // last of x points to first of y
      y.last→link = first;    // last of y points to first of x, for circular chains
      z.first = first;
      z.last = y.last
}
return z;
```