Solver: Eric Leonardo Lim

Email Address: elim023@e.ntu.edu.sg

1. (a)

(i) The output of the code is the sum of all elements in row $0..i$ which are positive integers and to the left of the first 0, if any, for all rows $i = 0..3$.

More formally, let x_i be the first column j in row i such that $a_{i,j} = 0$, and x_i is equal to the number of columns (which is 4) if there doesn't exists such j.
Let d_i be the sum of all $a_{i,j}$ in row i and column j such that $a_{i,j} > 0$ with $0 \leq j < x_i$.
Then, the output for each row $i = 0..3$ is $\sum_{k=0}^{i} d_k$.

Output:
3
10
12
20

(ii) Notice that f1 is a function that swaps the values of its parameters. This is because the code works like the following:

c = a + b
b' = c – b = (a + b) – b = a
a' = c – b' = (a + b) – a = b

a' and b' denote the new values of a and b and they are equal to b and a respectively.

Output:
2, 1
3, 1
2, 1

(b)

(i)
```
Line 5.   double sum = 0;     // floating point for computing average
Line 10.  while(s[i] != 0) {  // iterate through the array until a 0.
Line 18.  sum /= c;           // "/=" for divide operation
Line 19.  printf("%f\n",sum); // want to output average stored in sum.
```

(ii)
```
Line 6.   char *temp                 // lexicographically smallest string
Line 9.   for(i=0;i<5;i++)           // 0-based index
Line 10.  if(strcmp(temp,name[i])>0)// comparison of strings: strcmp()
Line 12.  printf("%s\n",temp);       // temp is string, *temp is char
```

For line 10, if operator > is used instead, it will return true if the **address** of the left string is larger than the address of the right string. As the strings are stored in a contiguous

memory block, it returns true if the index of the left string is larger than the index of the right string.

(c)

(i)
```
p = a;                          // pointer p points to a[0]
q = b;                          // pointer q points to b[0]
while(*p) p++;                  // after loop, p points to end of a
while(*q) *(p++) = *(q++);      // concatenate
*p = 0;                         // add '\0' string terminator
printf("%s\n",a);
```

2. (a)

(i) f2(3,5) = 5*f2(2,3)
= 5*5* f2(1,1)
= 5*5*5*f2(0,-1)
= 5*5*5*1
= 125

Output:
125

(ii) f3(5) = f3(4) + f3(3)
= f3(3) + f3(2) + f3(2) + f3(1)
= f3(3) + 1 + 1 + 1
= f3(2) + f3(1) + 1 + 1 + 1
= 1 + 1 + 1 + 1 + 1 = 5

Output:
f3(): 5

(b)

(i)
```
Line 2.   void f4(char *s, int *a, int *b)
Line 9.   void f4(char *s, int *a, int *b) {
Line 11.  if(*s>='A' && *s<='Z') (*a)++;
Line 12.  if(*s>='a' && *s<='z') (*b)++;
```

(ii) We need to come up with how the code should work. Otherwise, debugging the code will be more difficult. The code should work in the following way.

First, find the maximum character. Then, for all the characters in its left, copy them to their immediate right. Finally, assign the maximum character to be the first character of the string.

e.g. str = abdzc. Let max_character = z. Copy all characters in max_character's left to its immediate right, i.e. str = aabdc. Finally, assign max_character to be the first character of str, i.e. str = zabdc.

```
Line 4.   char str[100];      // size should be initialized for input
Line 13.  q = p+i;
```

```
Line 17.  while(q>p){
Line 18.  *q = *(q-1);
```

(c)

```
if(ar[0] == target) return 0;
if(rLookup(ar+1,n-1,target) >= 0)
      return rLookup(ar+1,n-1,target) + 1;
else return -1;
```

3. (a)

(i) `Line 8.  n * sizeof(int)`
`Line 13.  free(intArr)`

(ii) It may access memory space which is already allocated.

(b)

(i) 2,1

(ii) A 6-operation sequence is possible if and only if:

- Both push and pop operations are done exactly three times.
- Pop operations are done only when the stack is not empty, i.e. there exists at least an integer in the stack. We can have O as the next character in the sequence if the current number of O's is less than 3 and the current number of U's is more than current number of O's in the current sequence.

There are exactly 5 possible distinct such sequences:

UOUOUO -> 1,2,3
UOUUOO -> 1,3,2
UUOOUO -> 2,1,3
UUOUOO -> 2,3,1
UUUOOO -> 3,2,1

(iii) Queue's working principle is FIFO (First-In First-Out), whereas stack's working principle is LIFO (Last-In Last-Out).

(c)

```
while(p->next->next != NULL){    // there are at least 4 nodes
      p = p->next;
      size++;
}
size+=2;                         // because haven't count the last 2 nodes

tmp = head->next;
head->next = p->next;
head->next->next = tmp->next;
p->next = tmp;
tmp->next = NULL;
```

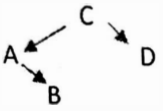
4. (a)

(i) Figure Q4(1) is not a BST because 14 is located in the left sub-tree of 13 and 14>13.

(ii) Let L_x be the set of all nodes in the left subtree of node x.
Let R_x be the set of all nodes in the right subtree of node x.
Figure Q4(2) is a BST because for all nodes x in the tree, x has at most 2 children and $\forall l \in L_x . \forall r \in R_x .\ l < x < r$,

(b)

(i)



(ii) B-A-D-C

(c)

```
int printParent(BTNode *node, BTNode *x){
      if(node == NULL) return 0;
      int flag = 0;
      if(node->left != NULL && node->left == x){
            printf("%d\n",node->item);
            flag = 1;
      }
      if(node->right != NULL && node->right == x){
            printf("%d\n",node->item);
            flag = 1;
      }
      return flag | printParent(node->left,x) | printParent(node->right,x);
}
```

For reporting of errors and errata, please visit pypdiscuss.appspot.com
Thank you and all the best for your exams! ☺