CS 2123 Data Structures

Fall 2016 – Final Exam – Saturday, Dec 10th (07:00am - 09:30am)

You have 150 min. + 5 min for the survey. Good luck.

This exam has 10 questions in 12 pages and an exit survey in page 13

You can use the 2-page C reference card posted in the class web page.

Name:…… Section:..... Score: ……./100 +(2pt bonus credit for survey)

1. (10pt) *[Pointers]* Trace the following code and show the changes in memory:

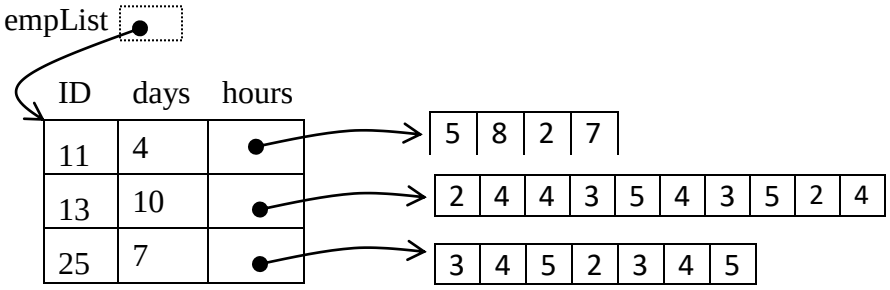
```
main() {
   int v1[] = {20, 30, 40}, *ptr;
   myfunc(&v1[1],  &ptr);
}
myfunc(int *p1, int **p2){
   *p2 = p1++;
   *p1-- = 80;
   **p2 = 60;
}
```

Var	Addr	content
v1 [0]	1004	
v1[1]	1008	
v1[2]	1012	
ptr	1016	
	1020	
p1	1024	
p2	1028	

2. (10pt) *[Strings]* Implement **`char *makeHTMLtag(char *s);`** which **creates a new string** by adding **`'<'`** in the beginning and **`'>'`** in the end of **s**. If no memory, return NULL. For example, **`makeHTMLtag`**`("abc xyz");` returns a new string as `"<abc xyz>"`

```
char *makeHTMLtag(char *s)
{       /* Assume standard libraries stdlib.h, stdio.h, string.h are included */
```

3. (10pt) *[Dynamic Memory Allocation]* Suppose somehow we have already created the below data structure. Now you are asked to write a function that can create a copy of such a given structure. Make sure your copy function deals with different number of employees as well as different number of days of each employee.




```
#include <stdio.h>
#include <stdlib.h>

typedef struct employee {
    int ID;
    int days;
    int *hours;
} employeeT;

int main(void)
{
  employeeT  *empList, *newEmpList ;
  int numEmp;


  numEmp = 3;
  empList =  CreateInitialEmpList(numEmp);
     /* suppose this function has been implemented for you and
        it has created the data structure shown above */


  newEmpList = CopyEmpList(empList, numEmp);
     /* You are asked to just implement this function in the next page */

}
```

```
employeeT *CopyEmpList(employeeT *oldList, int numEmp)
{
  employeeT *newList;
```

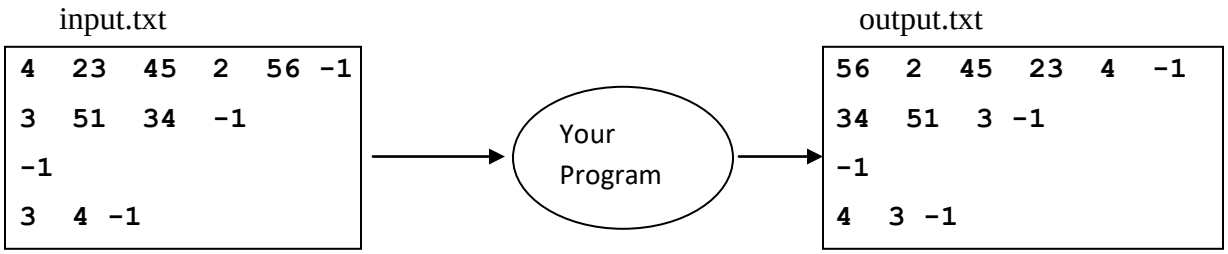
4. (10pt) *[Files and Stacks]* Suppose you are given the stack.h and its implementation stack.c. So you will just use it in your application (client) program below. Here are the important things in stack.h

```
#ifndef _stack_h
#define _stack_h
#include "genlib.h"

typedef void *stackElementT;
typedef struct stackCDT *stackADT;

stackADT          NewStack(void);
void              FreeStack(stackADT stack);
void              Push(stackADT stack, stackElementT element);
stackElementT     Pop(stackADT stack);
int               StackIsEmpty(stackADT stack);
#endif
```

Please note that the **`stackElementT`** is **`void *`**, so to store your data (e.g., int), you need to first allocate memory for your data (e.g., int), save your data at that address, and push that address onto stack. When you pop, you will get the address of your data. By dereferencing it, you can access your data.

Now you are asked to complete the application (client) program in the next page. This program simply reads the integer numbers in each line from an input file, AND prints the same numbers from each line into another file while **reversing their order**. Using stack library will make this reversing task easy!

For example,



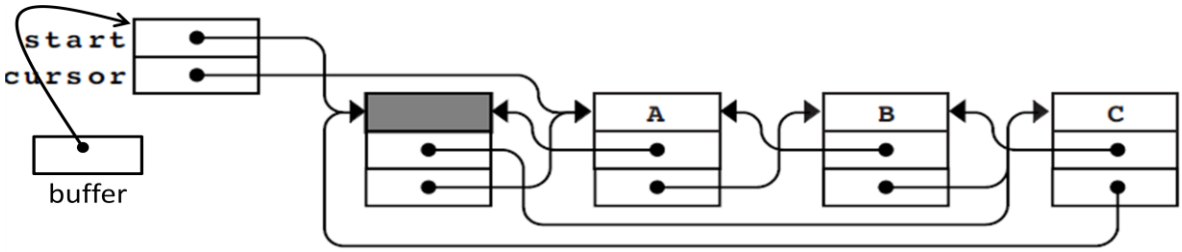
As seen in the above example, each line ends with a special value (-1) to indicate the end of line. After printing the numbers before -1 in reverse order, you should also print -1 and **`'\n'`** to indicate the end of each line in output file, too.

Make sure you check and release dynamic allocations and structures. Also close files...

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"
int main()
{
  FILE *infp, *oufp;
  StackADT stack;
  int num, *ptr;

  if((infp =  fopen("input.txt", "r"))==NULL) exit(0);
  if((outfp = fopen("outut.txt", "w"))==NULL){ flose(infp); exit(0); }
```

/* if needed, you can define other variables here */

5. (10pt) Recall the buffer ADT, `buffer.h` and suppose we consider its implementation based on the **circular double link list (CDLL)** representation. For **A | B C**, it can be visualized as follows



You are asked to implement the following function which removes the character AFTER the cursor if the cursor is not at the end. After calling it for the above case, the buffer will have: A **|** **C.**

/* CDLL-imp-buf.c */

```
#include "buffer.h"

typedef struct Dcell {
    char ch;
    struct Dcell *prev;
    struct Dcell *next;
} DcellT;

struct bufferCDT {
    DcellT *start;
    DcellT *cursor;
};

bufferADT NewBuffer(void)
{
 bufferADT buffer;
 DcellT *dummy;
 buffer = New(bufferADT);
 dummy = New(DcellT *);

 buffer->start =
 buffer->cursor =
 buffer->start->prev =
 buffer->start->next = dummy;
 return (buffer);
}

/* implementations of  other functions */
```

```
void DeleteCharAfterCursor(bufferADT buffer)
{
   DcellT *cp;
```

6. (10pt) *[Binary Trees]* Suppose we added a new field (called `numC)` into the binary tree structure, as shown below. The new field represents the number of children in each node. So it will be 0, 1, or 2. But, our current insert function does not set this field correctly.

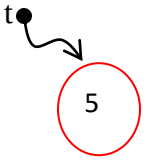
   So, you are asked to write a function `void SetNumberOfChildren(nodeT *t);` which sets the this field correctly for every node in a given binary tree.

```
typedef struct node  {
   int key;
   int numOfChildren;
   struct node *left,  *right;
}  nodeT, *treeT;

void SetNumberOfChildren(nodeT *t);
{
```

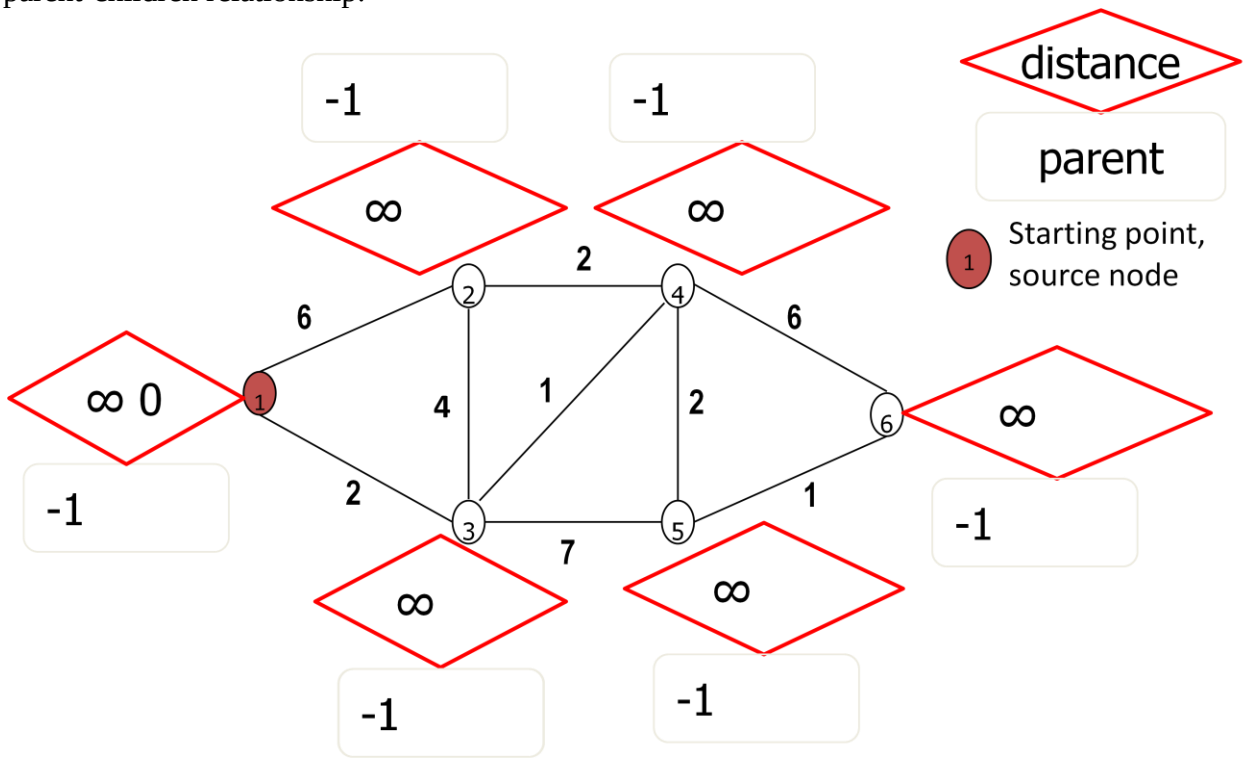
7. (10pt) *[Balanced Binary Search Tree (BST) and AVL]* Suppose you get the following numbers one at a time and insert into the BST using AVL balancing algorithm.

5 10 30 17 24 4 3

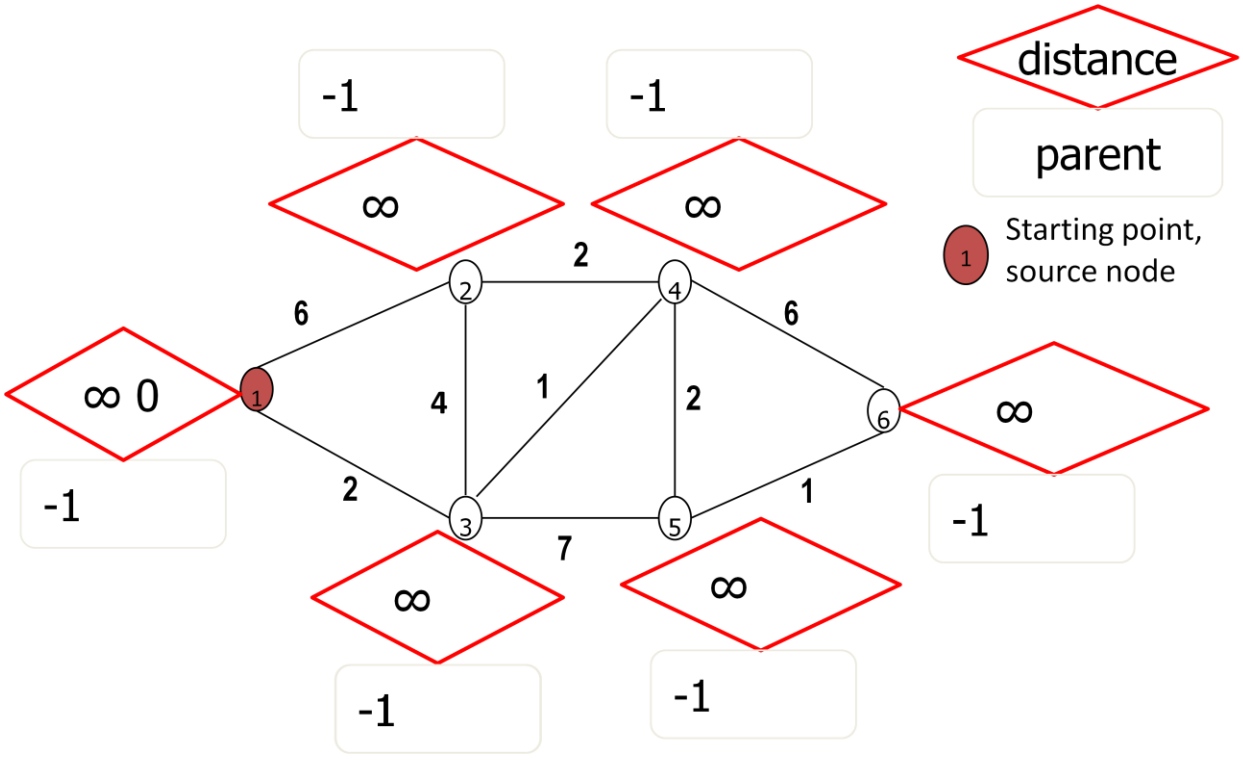
Show how the tree grows as you insert each number. If there is no rotation, keep inserting the new node into the existing tree. But if there is a rotation, please re-draw the tree after each rotation so that we can clearly see the type of rotation(s) you did.



8. (10pt) Using the below graph, show how shortest path algorithm (known as Dijkstra's algorithm) finds the shortest paths starting from **NODE 1** to every other node. Don't just inspect the graph and give solutions! Instead follow the Dijkstra's algorithm by showing all the changes in distance and parent labels, as we did in class. Then use solid arrows to show parent-children relationship.

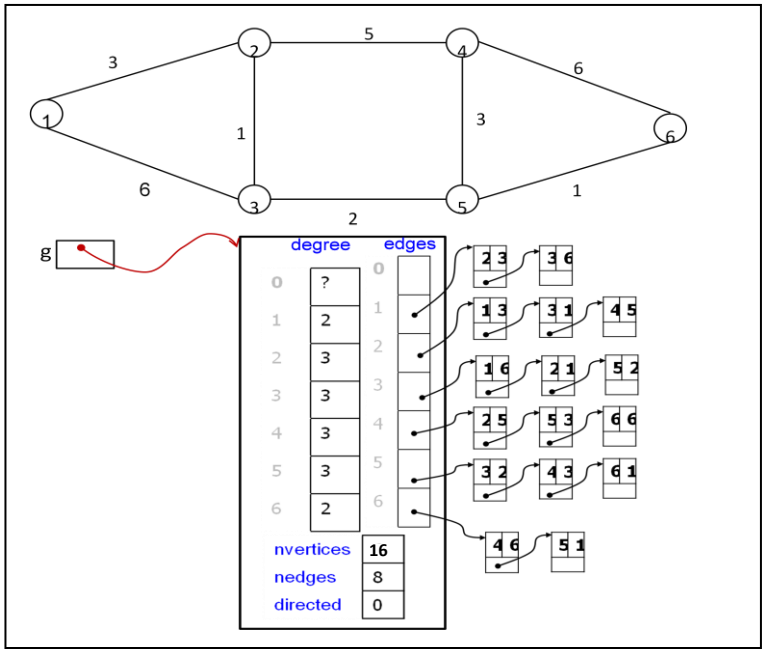


Here is another copy. If you make too many mistakes in the above one, you can use this one…..



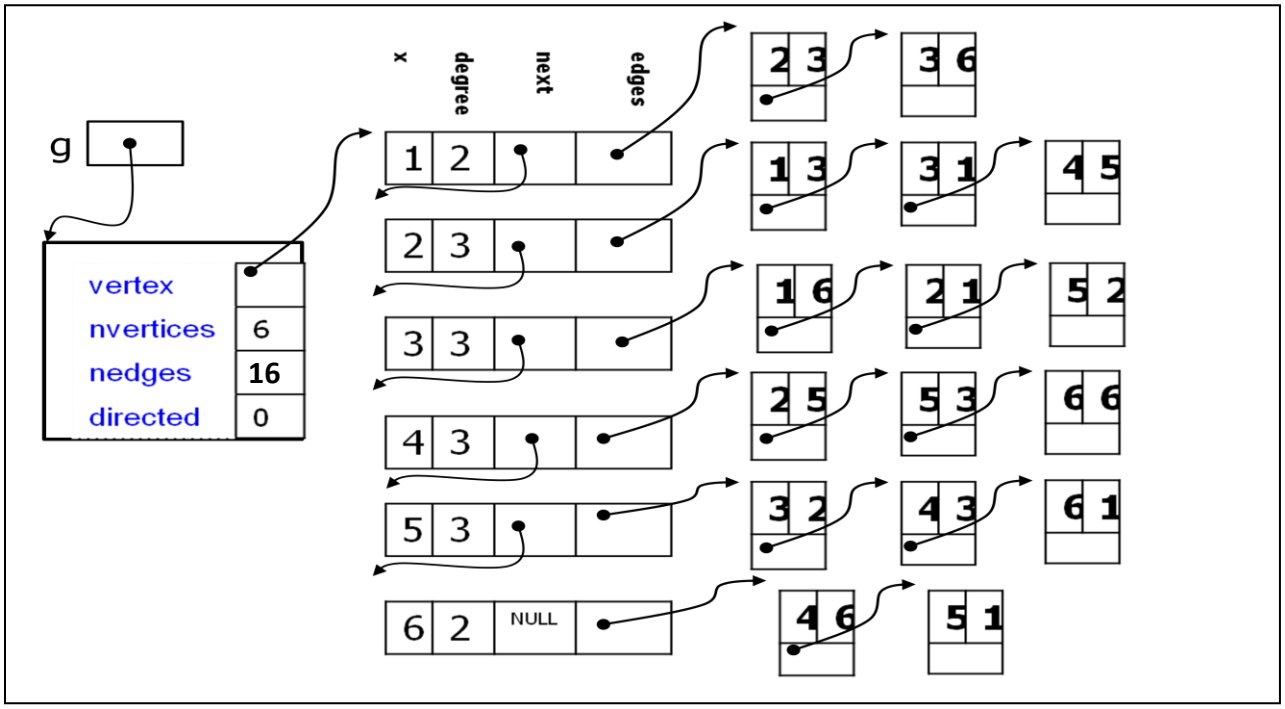
9. (10pt) Recall that we used the following structures to represent a graph using **adjacency list.**

```
#define MAXV 6
typedef struct edgenode {
    int y;
    int w;
    struct edgenode *next;
} edgenodeT;
typedef struct {
    edgenodeT *edges[MAXV+1];
    int degree[MAXV+1];
    int nvertices;
    int nedges;
    bool directed;
} graphT;
graphT *g;
```



One of the disadvantages of the above representation is that the maximum number of nodes is fixed to MAXV. So we waste some spaces when we don't have that many nodes or we cannot use this program if we have more nodes than MAXV. Accordingly, we want to remove fixed size array and use a link list to dynamically store nodes, their IDs, degrees etc. So we need to modify `graphT` and define a new cell structure (say `nodeT`) to store nodes in a linked list. We like to keep `edgenodeT` as is.

We give the new form of `graphT` and `nodeT` structures in the next page such that we can conceptually represent the above same graph as follows:



typedef struct node {
int x; // vertex id
int degree;
struct node *next;
edgenodeT *edges;
} nodeT;
	typedef struct {
	nodeT *vertex;
	int nvertices;
	int nedges;
	bool directed;
	} graphT;

NOW you are asked to implement `int delete_link(graphT *g, int i, int j);` based on the new representations/structures defined above. It basically deletes the edge (i, j) from the graph and returns 1. If link (i,j) does not exist, the function simply returns 0.

```
int delete_link(graphT *g, int i, int j);

{
   edgenodeT *pe, *prev;
   nodeT *pn;
```

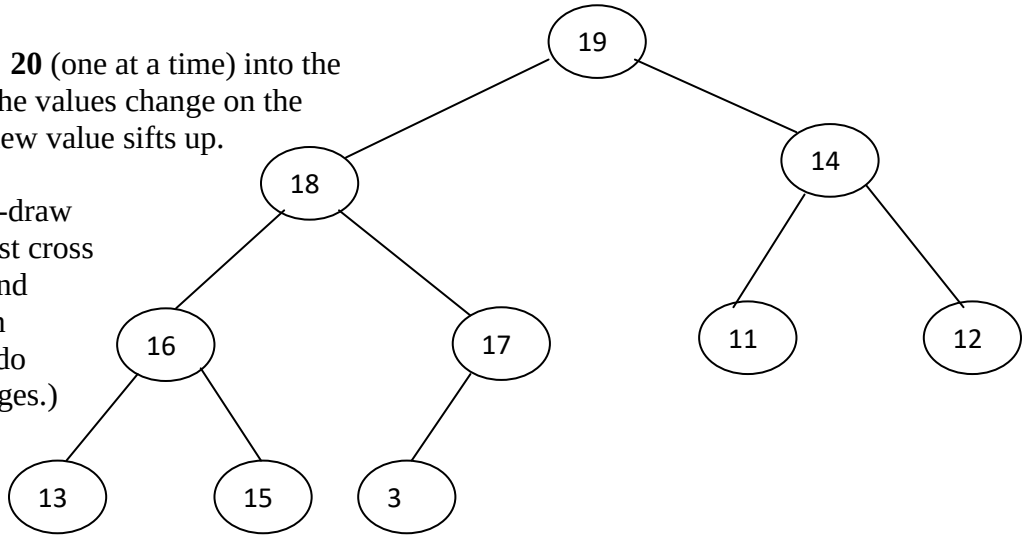
10. (10pt) *[Heaps]* You are given the following MAX HEAP and its visualization as a tree strucutre with the current number of elements **n** = 10.

	0	1	2	3	4	5	6	7	8	9	10	11
heap	**19**	**18**	**14**	**16**	**17**	**11**	**12**	**13**	**15**	**3**		

a. (4pt) Insert **24** and **20** (one at a time) into the heap and show how the values change on the tree structure as the new value sifts up.

(You don't need to re-draw the tree everytime, just cross the previous values and put the new values on the same tree as you do the necessary exchanges.)

We need to see these changes!



b. (1pt) Then give the final version of the heap array, basically map the numbers from the above tree structure to the heap array.

	0	1	2	3	4	5	6	7	8	9	10	11
heap												

c. (5pt) Complete the missing part of the following siftup function.

```
void siftup(int heap[], int r, int n) {
  int parent, tmp;
  parent = PARENT(r,n);
  while(parent >= 0){




    parent = PARENT(r,n);
  } // end of while
}
```

Exit Survey (2pt bonus credit) *[if applicable, Circle just one option per question]*

1. Did you solve the sample final exam problems provided at the class web page?

   [1] No, None. [2] Yes, Some . [3] Yes, Most. [4] Yes, All of them

2. If **Yes**, did you solve them with Tutor, TAs, both, or by yourself?

   [1] Myself only [2] Tutor only [3] TAs only [4] Tutor, TAs, myself

As you know, The Department had provided Tutor and Common TAs to help you. Please answer the following questions based on your experience with Tutor and common TAs.

Please answer the followings IF you attended any **tutoring** sessions and got some help from **our Tutor (**Javier),

3. How **knowledgeable** was the Tutor about the subject matter of the course?

   [1] Not at all knowledgeable [2] Slightly... [3] Moderately ... [4] Quite knowledgeable

4. How **clearly** did the Tutor present material?

   [1] Not at all clearly [2] Slightly ... [3] Moderately ... [4] Quite clearly

5. Overall, how would you rate the **usefulness** of **tutoring** in your learning the subjects?

   [1] Not at all useful [2] Slightly ... [3] Moderately ... [4] Quite useful

6. Throughout the semester, how many times did you get any help from **the other common TAs?** *[Consider TAs other than our own course TAs (**Maryam or Kavita**)]*

   [1] None. [2] 5 times or less. [3] 10 times or less. [4] More than 10 times.

7. What **type of help** did you get the most from **other** common TAs in the main CS lab?

   [1] Help with Programming assignments,

   [2] Help with Solving sample exam questions,

   [3] Help with System problems such as Login, using Linux, VDI, SSH etc.

8. How **easy** did you find it to meet with **other** common TAs in the main CS lab?

   [1] Not at all easy [2] Slightly ... [3] Moderately ... [4] Quite easy

9. How **knowledgeable** were **other** common TAs about the subject matter of the course?

   [1] Not at all knowledgeable [2] Slightly... [3] Moderately ... [4] Quite knowledgeable

10. How **clearly** did **other** common TAs present material?

    [1] Not at all clearly [2] Slightly ... [3] Moderately ... [4] Quite clearly

11. Overall, how would you rate the **usefulness** of **other** common TAs in your learning the subject matter and passing this course?

    [1] Not at all useful [2] Slightly ... [3] Moderately ... [4] Quite useful

Please use the next page if you like to provide more feedback about Tutoring or Common TAs (e.g., its weaknesses, strengths, how to make it more effective etc.)