Name / ID (please PRINT) **Sequence #:_____** **Seat Number:____**

_____ _____

CS 2123.001 Data Structures

Spring 2018 – FINAL

CS 2123.001 Monday 7-May 9:45 AM - 12:15 PM

- This is a **closed book/note** examination. But *You can use the reference card(s) given to you.*
- This exam has 9 questions in 12 pages. Please read each question carefully and answer all the questions, which have 100 points in total. Feel free to ask questions if you have any doubts.
- **Partial credit will be given, so do not leave questions blank**.

Question	Topic	Possible Points	Received Score
1	Pointers (tracing a program)	10	
2	String processing/manipulation	10	
3	Dynamic Memory allocation	10	
4	Application of ADT or addition/modification to ADT (One of StackADT, QueueADT, BufferADT)	10	
5	Trees - Recursion	10	
6	Trees - AVL	10	
7	Graphs - Shortest path tracing	10	
8	Graphs - Coding	15	
9	Heaps	15	
Survey Bonus credits	After Final Exam, I will e-mail you a link to complete a survey about Tutoring and zyBook. If you complete it, you will earn 1-point bonus credit that will be added to your overall total.		
Total		100	

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

```
typedef struct cell {
   int x, y, z;
} cellT;
main() {
   cellT v[2]={{10,20,30},{40,50,60}};
   cellT *ptr;
   myfunc(&v[0],  &ptr);
   ptr->x = 75;
   ptr++;
   ptr->z = 65;
}
void myfunc(cellT *p1, cellT **p2){
   *p2 = p1++;
    p1->y = 85;
   (*p2)->y = 95;
}
```

Var	Addr	content
v[0].x	1004	10
v[0].y	1008	20
v[0].z	1012	30
v[1].x	1016	40
v[1].y	1020	50
v[1].z	1024	60
ptr	1028	
	1032	
	1036	
p1	1040	
p2	1044	

2. (10pt) *[Strings]* Suppose you are given a URL which has the following format:

protocol://hostname/file_info

Now you are asked to write a function that can extract the protocol and hostname parts as **dynamically created** two new strings and return them. Here is the prototype for that function:

```
void get_protocol_hostname(char *url, char **pp, char **ph);
```

For example, after

```
char *url = "http://www.cs.utsa.edu/~korkmaz/index.html";
char *p, *h;

get_protocol_hostname(url, &p, &h);
```

p points to **"http"** and **h** points to **"www.cs.utsa.edu"**

Assume that the given URL is a valid string and has the format as specified above. So you **don't** need to check for the correctness of the URL format. But if anything else goes wrong (e.g., no memory) in the function, make sure **p and h** will be **NULL.**

```
/* give your solution to this problem in the next page */
```

```
void get_protocol_hostname(char *url, char **pp, char **ph)
{       /* You are NOT allowed to use any standard library functions
           except  strlen() and malloc()  if needed  */
        char *p, *h;
        int plen, hlen, i;
        *pp = NULL;  *ph = NULL;
        if (url == NULL) return 0;
```

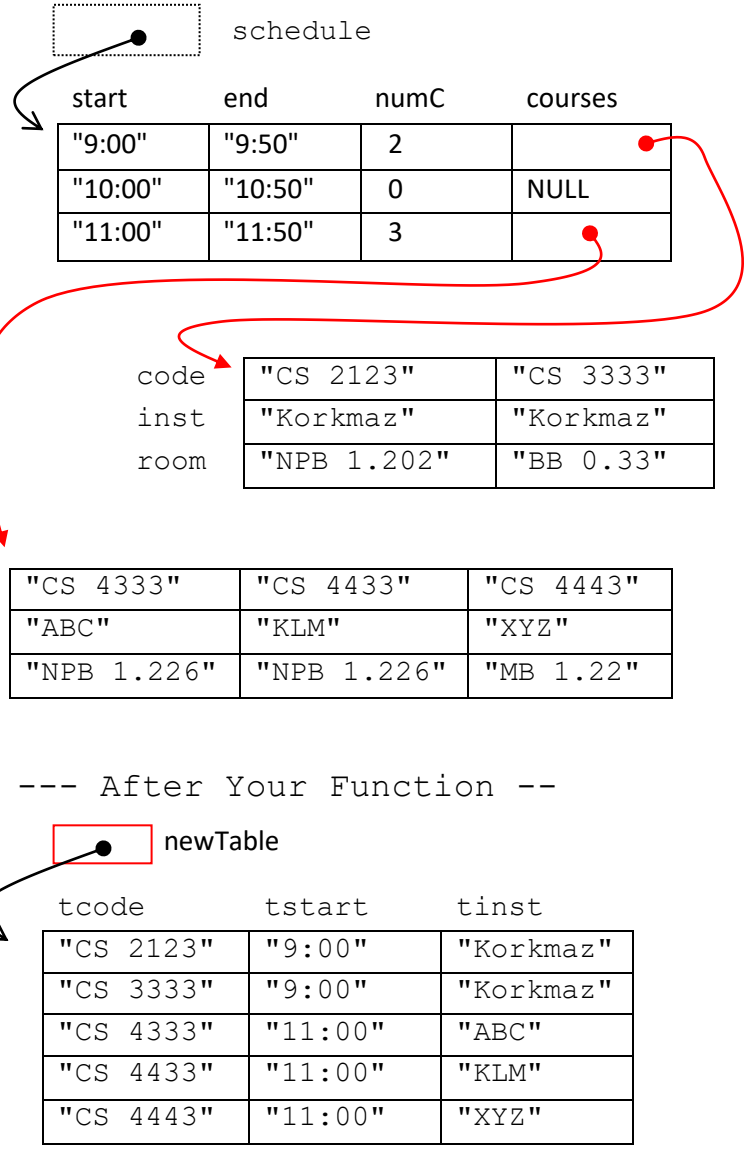
3. (10pt) *[Dynamic Memory Allocation]* Suppose somehow we have created the below data structure pointed by `schedule`. Now you are asked to write a function that can create a dynamic array pointed by `newTable` (containing course code, start time. and instructor) as shown below. Make sure your function deals with different number of courses (`numc`), too!

```
typedef struct courseT {
    char code[8];
    char inst[20]; // instructor
    char room[10];
} courseT;

typedef struct scheduleT {
    char start[6];
    char end[6];
    int numc; // number of courses
    courseT *courses;
} scheduleT;

typedef struct tableT {
    char tcode[8];
    char tstart[6];
    char tinst[20]; // instructor
} tableT;

int main(void)
{
  scheduleT  *schedule;
  int numOfTimeSlots = 3;
  tableT *newTable;
```



--- After Your Function --

/* Suppose somehow we created the above data structure pointed by `schedule` */

/* You are asked to just implement the following function in the next page */

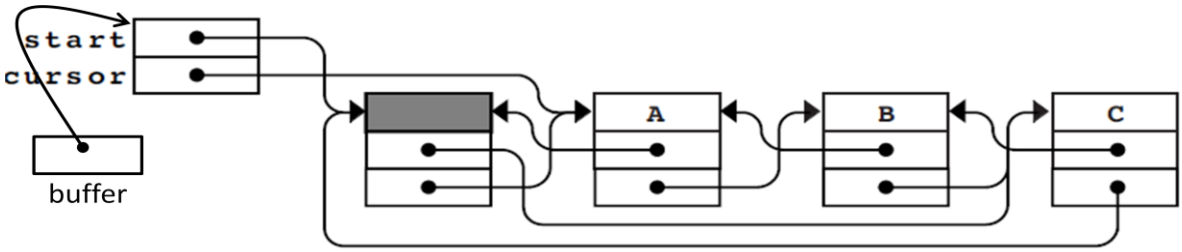
`newTable = Create_CSI_Table(oldSchedule, numOfTimeSlots);`

/* You are allowed to use any standard library functions that you may need */

}

```
tableT *Create_CSI_Table(scheduleT *sch, int numTS)
{
  tableT *newT;
```

4. (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 consecutively repetitive characters AFTER the cursor if they are the same as the character BEFORE the cursor. If the cursor is at the end of the buffer or at the beginning of the buffer, then there is no action to take.
After calling this function for the above case, the buffer will still have: **A | B C**
But if buffer has **A | A A A A B A A C** then buffer will have: **A | B A A C**

/* CDLL-imp-buf.c */

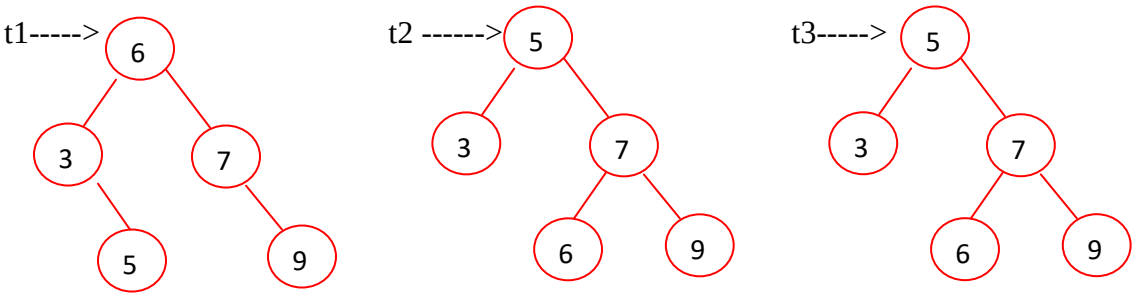
```
#include "buffer.h"
typedef struct Dcell {
    char ch;
    struct Dcell *prev;
    struct Dcell *next;
} DcellT;
struct bufferCDT {
    DcellT *start;
    DcellT *cursor;
};

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 DeleteRepeatedChars(bufferADT buffer)
{
    DcellT *cp;
```

5. (10pt) *[Binary Search Tree (BST)]* You are asked to implement the following function: `int areSameBSTs(nodeT *t1, nodeT *t2);` which checks if the given two BSTs have the same values and shape.
For example, it returns 1 for (t2, t3); while it returns 0 for (t1, t2) because they have different shapes even though they have the same values. It should also return 0 for the trees having the same shape but different values.




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

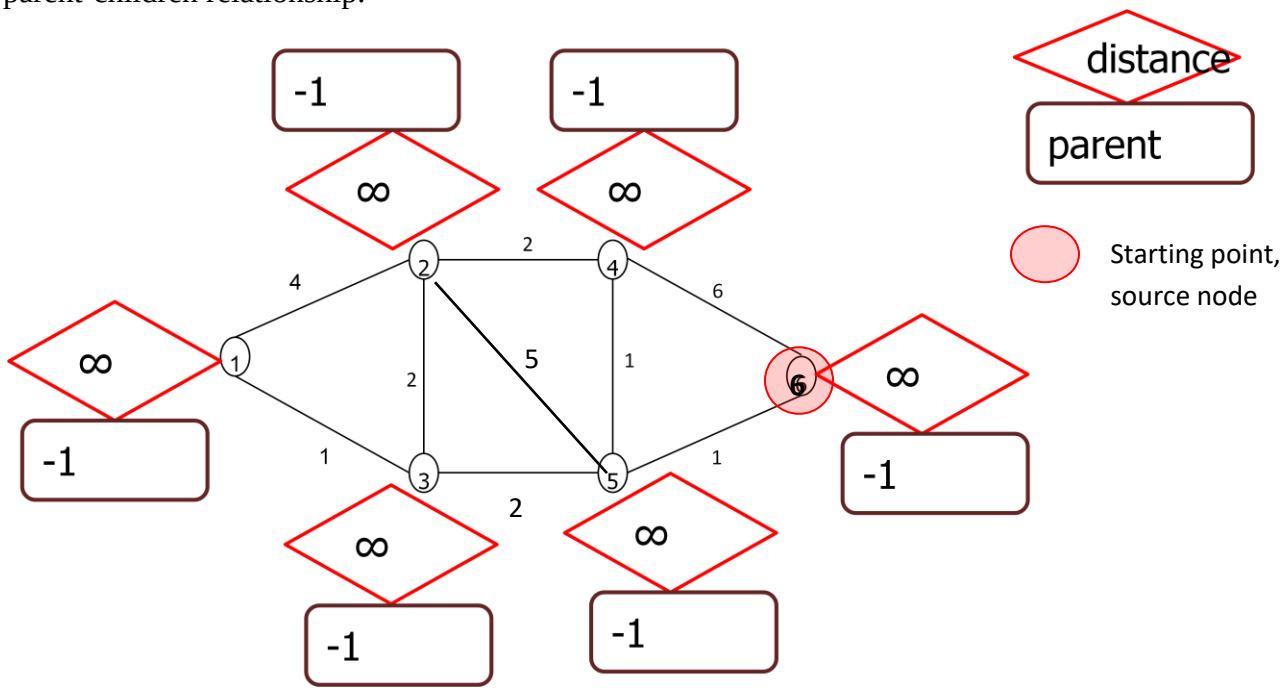
```
int areSameBSTs(nodeT *t1, nodeT *t2);
{
```

6. (10pt) *[BST and AVL]* Insert the following values into an AVL tree one by one. Initially the tree is empty. No need to re-draw the tree after each insertion. But after a rotation, you must re-draw the tree.

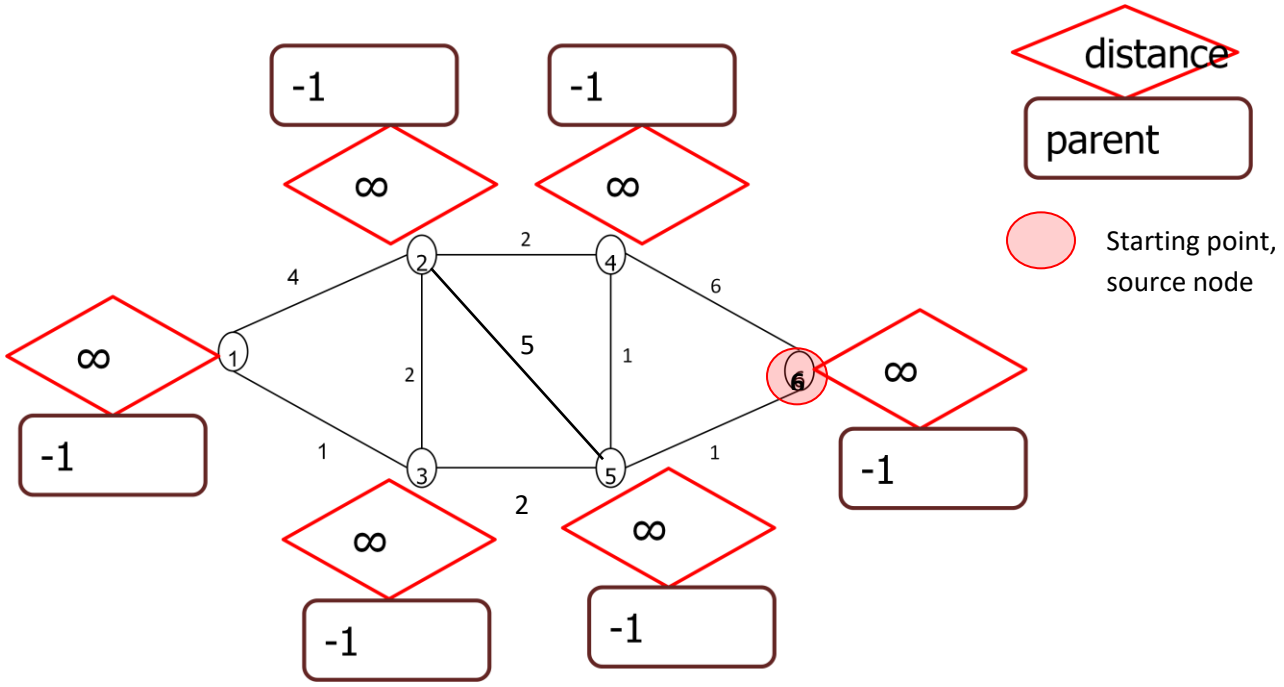
4, 3, 6, 7, 11, 5, 8, 9

t 

7. (10pt) Using the below graph, show how shortest path algorithm (known as Dijkstra's algorithm) finds the shortest paths starting from **NODE 6** 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…..



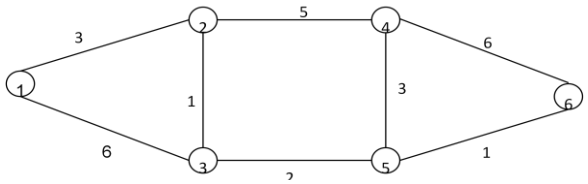
8. (15pt) Recall the graph structure we used in class with a fixed size array of pointers to the **adjacency list** of each node. One of the disadvantages of that representation is that the maximum number of nodes is fixed to MAXV. So either we waste some spaces when we don't have that many nodes, *OR* we cannot use this program if we have more nodes than MAXV. In response, we decided to remove the fixed size array and use a linked list to dynamically store node information (node IDs, degree, next, edges). Accordingly, as shown below, we modify `graphT` structure and define a new cell structure (say `nodeT`) to store nodes in a linked list while keeping `edgenodeT` as is to represent adjacency lists.

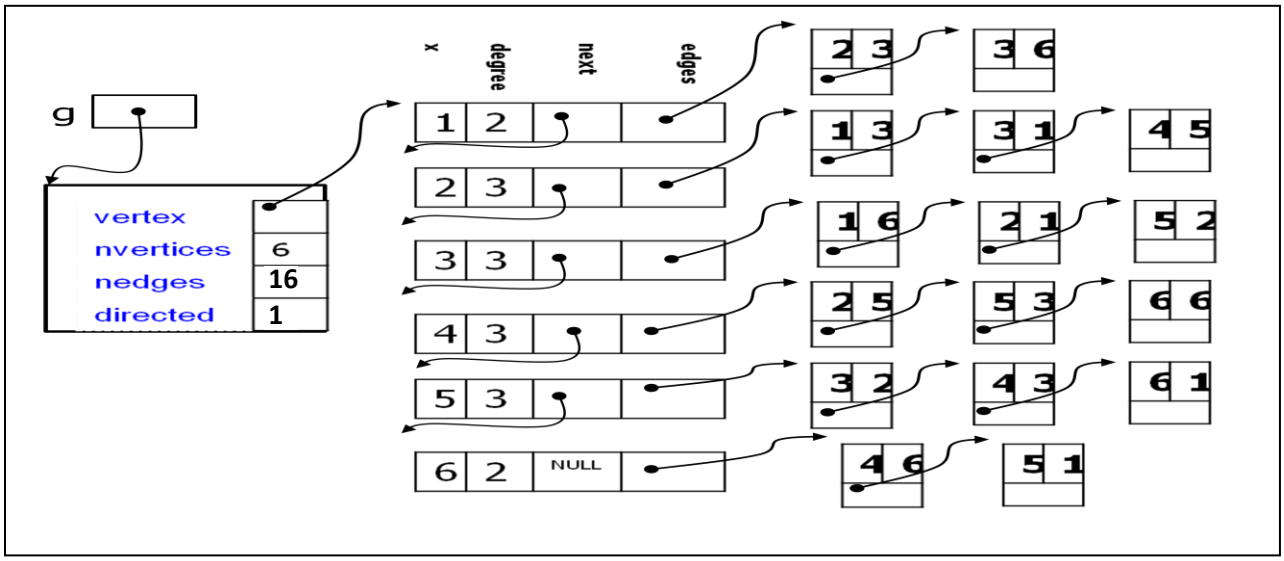
```
typedef  struct  node {
   int x; // vertex i
   int degree;
   struct  node  *next;
   edgenodeT *edges;
} nodeT;
```

```
typedef struct {
   nodeT *vertex;
   int nvertices;
   int nedges;
   bool directed;
} graphT;
```

```
typedef struct edgenode {
   int y;  // vertex j
   int w;
   struct edgenode *next;
} edgenodeT;
```

Here is how the graph on right will conceptually be represent with the new structures above.





NOW you are asked to use the above structures and delete a directed edge (i,j) from the the graph. [Don't worry about edge (j, i)!]

Function prototype is **`void delete_link(graphT *g, int i, int j );`**

- [10pt] If edge (i,j) is not in the graph, then there is no action to take. Otherwise, remove it from the graph.
- [5pt] After removing edge (i,j), if the degree of node i goes to 0, then remove the node i, too. (The same thing can be done for node j, **but** to simplify the implementation, don't worry about node j and leave node j as is)

```
void delete_link(graphT *g, int i, int j)
{
  nodeT     *pn, *prevN = NULL;
  edgenodeT *pe, *prevE = NULL;

  if (!g) return 0;
```

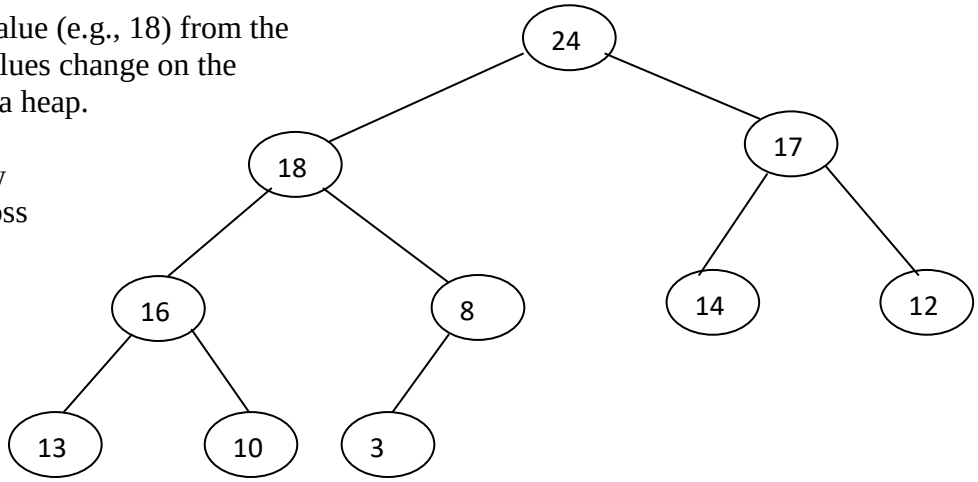
9. (15pt) *[Heaps]* You are given the following MAX HEAP and its visualization as a tree strucutre with the current number of elements **n** = 10.

	0	1	2	3	4	5	6	7	8	9	10	11
heap	24	18	17	16	8	14	12	13	10	3		

a. (6pt) Remove a **given** value (e.g., 18) from the heap and show how the values change on the tree structure to keep it as a heap.

(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) After removing 18, 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. (8pt) You are asked to implement a function that removes a given value from the heap if that value is in the heap. Assume that the following functions and the macros in them are available for you to use if needed:

```
void siftdown(int heap[], int r, int n); and
void siftup(int heap[], int r, int n);
```

```
void remove_given_val(int heap[],  int *n, int val) {
  int tmp, r;

  if (*n <=0 ) return;
  /* Hint: first search for the given value in heap array */
```