CS 2123 Data Structures

Fall 2016 – Midterm 2 -- Oct 27, 2016
You have 75 min. Good luck.
This exam has 5 questions in 8 pages
You can use the 2-page C reference card posted in the class web page.

Name:……........……………….........…   Section:...............   Score: ……./100

SURVEY (2pt bonus credit)

A. Within the last 5 weeks (after midterm 1)
How many TUTORING sessions did you attend?:__________
How many times did you get help from COMMON TAs in the Main CS lab?: __________
How many times did you get help from the PROFESSOR during his office hours?: ______

B. Which one you do you think is more useful; thus should be increased? (circle one)

NONE(0)   TUTORING(1)   COMMON_TA(2)   BOTH(3)

C. What other help do you think the Department should provide?

____

____

1. (20 pt) Review Questions:
   a. (4pt) Suppose one of your friends tries to implement a function that can dynamically create a new copy of a given string. He/she came up with the following one:

```
char *CopyString(char *s)
{
  char *ns;
  ns = (char *) malloc(sizeof(s));
  if(ns==NULL) return NULL;
  strcpy(ns,s);
  return ns;
}
```

It compiles OK and works sometimes (when the original string is very short). But most of the time (when the original string is longer) it fails and gives a segmentation fault. What is the problem, why does it work sometimes, and how would you fix it?

Give your answer in next page.....

`The problem and the reason it may work sometimes(2pt):`

`The fixed version (2pt):`

b. (4pt) Suppose we want to compute the **maximum** of the integer values in an array using a function: `int arr_max_recursive( int arr[], int n );` This function can easily be implemented as an iterative function. **However**, you are asked to think about a recursive strategy and implement `it` as **a recursive function**.

`int arr_max_recursive( int arr[], int n )`
`{` *// Must be* ***recursive*** *(No credit will be given for an iterative implementation*!)

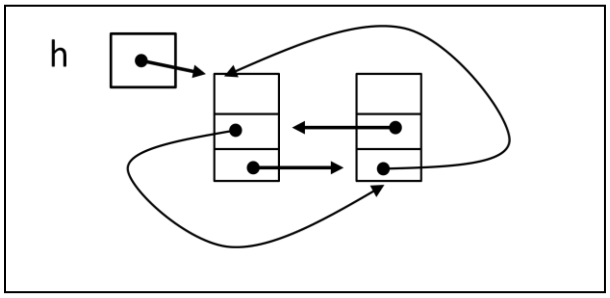
c. (4pt) In the following loops, first count the number of operations (i.e., find how many lines will be printed out) when N is 32. Then give the computational complexity using big-O notation for any value of N. (justify your answers)

`int i, j, N=32;`	Number of lines printed when N=32	big-O notation

`for(i=N; i >= 1; i--)`
`  for(j=1; j <= i; j++)`
`     printf(" line1\n");`		

`for(i=2; i <= N; i = i * 2)`
`  for(j=N; j >= 2; j = j / 2)`
`     printf(" line2\n");`		

d. (4pt) Suppose we want to create a circular doubly linked list, as shown below.



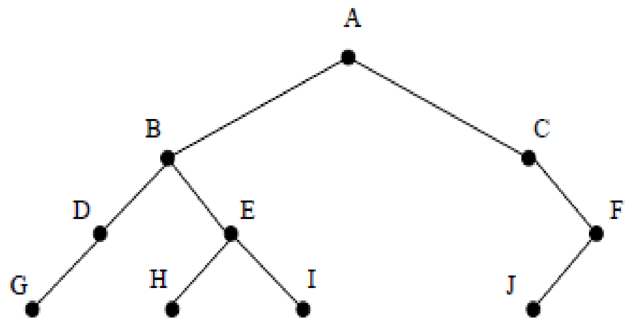
```
typedef struct DcellT {
    int val;
    struct DcellT *prev;
    struct DcellT *next;
} DcellT;

DcellT *h, *newcell;
```

For this, we declare the cell structure given above. Now you are asked to complete the below code segment to inset a new cell. New cells will be inserted in front of the list. Suppose we already created a new cell pointed by `newcell` and initialized it..

```
if (h==NULL){
   h=newcell; newcell->next = newcell->prev = newcell;
} else {
```

e. (4pt) List the sequence of nodes visited according to the following traversals:

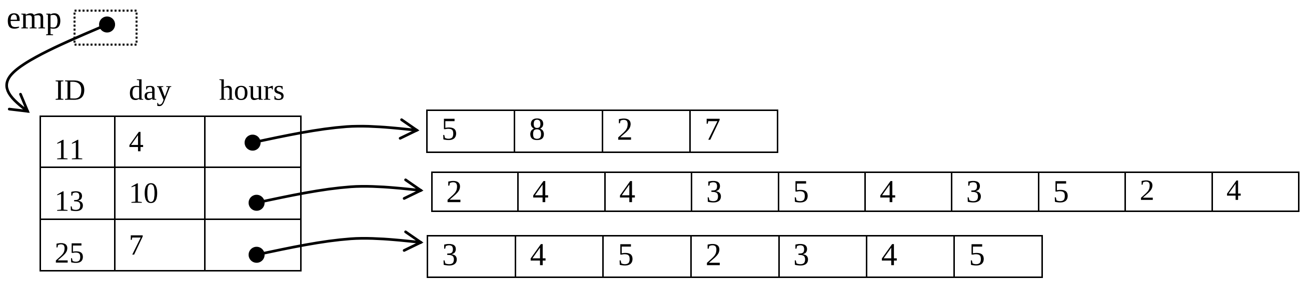


Preorder:
Inorder
Postorder:

2. (20pt) Suppose we have an employee data file (say **`emp.txt`**) containing the followings: first an integer representing the number of employees (lines) in the file, then we have that many lines where each line contains: *employee ID*, *how many days he/she worked*, and *how many hours he/she worked in each day*. For example, here is a sample **`emp.txt`** file with three employees (an actual file may have more lines or days, so your program should be general enough to work with other files containing different number of employees or days)

```
3
11    4     5 8 2 7
13   10     2 4 4 3 5 4 3 5 2 4
25    7     3 4 5 2 3 4 5
```

You are asked to complete the following program that can read emp.txt file and just create a dynamic structure to store all the above information in the memory, as shown below.
`/* we will do something with the structure later */`



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

typedef struct employee {
    int ID;
    int day;
    int *hours;
} employeeT;
```

```
int main(void)      /* give your answer for question 2 here */
{
  FILE *infp;
  employeeT  *emp;
  int i, j, numEmp;       /* if needed you can declare other variables here */

  if ((infp = fopen("emp.txt", "r"))==NULL){
     printf("Input file cannot be opened\n");
     return -1;
  }
  fscanf(infp, "%d",&numEmp);
```

3. (20pt) Recall the stack ADT. Followings are the important things from its interface stack.h

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

typedef void *stackElementT;
typedef struct stackCDT *stackADT;

void Push(stackADT stack, stackElementT element);
stackElementT Pop(stackADT stack);
#endif
```

Please note that the `stackElementT` is `void *.` But you need to push/pop `double` values in your application. In this case, you need to first dynamically allocate memory space for your data, save your data at that space, and push its address. When you need the data, you can pop its address, dereference it and get the value, then free the memory space.

Instead of doing these things whenever needed, you can simply implement the following two functions that can push and pop a double value as described above, then you can call them when needed. (Don't forget to check if memory is allocated or not in push, and to free it in pop)

```
void myPushDouble(stackADT stack, double value) // 14pt
{
     double *ptr;









}
double myPopDouble(stackADT stack)  // 6pt
{
   double tmp, *ptr;
```

4. (20pt) Recall the list ADT. Suppose its interface is extended as below. Mainly we include the following prototype into `list.h`: **`void RemoveOddValues(listADT a);`** which removes/deletes the odd values from the list. So at the end list will contain only even values. Implement it based on the structures given below.

```
#ifndef _list_h_
#define _list_h_

typedef struct listCDT *listADT;

listADT NewList();
void list_insert_sorted(listADT a, int val);
void list_insert_unsorted(listADT a, int val); // add val to the end
void RemoveOddValues(listADT a);
#endif
```

**/* list.c
using linked list */**
 	
/* suppose standard and book libraries are included too */
 	
#include "list.h"
 	
typedef struct point {
   int x;
   struct point *next;
} myDataT;
 	
struct listCDT {
      myDataT *start;
      myDataT *end;
};
 	
listADT NewList()
{
      listADT tmp;
      tmp = New(listADT);
      tmp->start = NULL;
      tmp->end = NULL;
      return(tmp);
}
 	
/* implementations of other functions */
	**void RemoveOddValues(listADT a)
	{**

5. (20pt) Recall the set ADT which had the same interface `set.h` but two different implementations (i.e., list and array). Suppose we like to add a new function to check if set A is the subset of set B.

   Note: If every member of set *A* is also a member of set *B*, then *A* is said to be a *subset* of *B*, written $A \subseteq B$ (also pronounced *A is contained in B*).

   Add the following prototype to `set.h`: `int isSubset(setADT A, setADT B);` which returns 1 if A is subset of B (i.e., every member of set A is also member of set B); otherwise, returns 0. You are asked to implement this function under link list implementation.

   Assume that both sets are sorted!

```
/* setLinkedListImp.c   */

#include "set.h"

typedef struct cellT {
    setElementT val;
    struct cellT *next;
} cellT;

struct setCDT {
    cellT *start;
    cellT *end;
};

setADT setNew(void)
{
 setADT set;
 set = New(setADT);

 set->start = NULL;
 set->end = NULL;
 return (set);
}

/* implementations of  other functions */
```

```
int isSubset(setADT A, setADT B
{
```