CS 2123-001-002 Data Structures

Fall 2016 – Midterm 1 -- September 22, 2016
You have 75 min. Good luck.
You can use the 2-page C reference card posted in the class web page.

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

Background Survey (3pt bonus credit)

A. Please complete the below table for the computer-programming-related courses that you have taken before Fall 2016.

Programming courses	TAKEN AT UTSA? If **YES**, give the instructor's name and term (e.g., Clark, Tosun, Robinson, Sherette or ... in Summer 2016).	If **NO**, but if you have taken equivalent courses from a **different** school, give the **school name** and the **language** used.
CS 1063 Intro to Comp Prog I (in Java) When is it taken:		
CS 1713 Intro to Comp Prog II (in C) When is it taken:		

B. How would you evaluate your programming skills and background? (circle one)

5: excellent, received As in all Intro programming courses.

4: good, received A-B in Intro programming courses.

3: average, received Bs in all Intro programming courses

2: fair, received B-C in Intro programming courses.

1: just passed, received Cs in all Intro programming courses.

C. How many tutoring sessions did you attend? (Circle one): 0 1 2 3
If you worked with tutor at all, how would you evaluate the usefulness of the tutoring program?

○ Definitely. ○ Somewhat. ○ I don't know. ○ Not really. ○ Definitely not.

D. How many times did you get help from other common TAs in the Main CS lab (except our TAs Kavita and Maryam)? (Circle one): 0 1 2 3 4 5 6
If you got any help, how would you evaluate the usefulness of the common TA program?

○ Definitely. ○ Somewhat. ○ I don't know. ○ Not really. ○ Definitely not.

1. (20 pt) Implement a function `char *head_tail_trim(char *s1, int h, int t);` which copies the characters (except the first `h` character and last `t` characters) from the given string `s1` into a dynamically created new string and returns the pointer to this new string.

   If `t+h` is greater than the length of the string, or `h` or `t` is less than 0, return NULL

   For example:
   After `s = head_tail_trim("ABCDEFG", 2, 3);` `s` should be pointing to `"CD"`
   After `s = head_tail_trim("ABCDEFG", 0, 3);` `s` should be pointing to `"ABCD"`
   After `s = head_tail_trim("ABCDEFG", 3, 4);` `s` should be pointing to `""`
   After `s = head_tail_trim("ABCDEFG", 5, 4);` `s` should be NULL.
   After `s = head_tail_trim("ABCDEFG", -2, 3);` `s` should be NULL.

   Suppose standard libraries are included. So, standard library functions can be used if needed.

```
char *head_tail_trim(char *s1, int h, int t)
{    /* you can use either pointer or array notation */
```

2. (20 pt) **Trace** the following program, **show how values change** in memory, and **give the output.**

```
#include <stdio.h>
main()
{
  int x=9, y=3, z[4]={0}, *p1, **p2;

  p1 = &z[3];

  *p1-- = 8;

  p2 = &p1;

  **p2 = 7;

  *--p1 = 5;

  printf("%d %d %d \n",
         p1, *p1, &p1);

  printf("%d %d %d %d \n",
         p2, *p2, **p2, &p2);

  x = f1(&y, **p2, *p2+1, &p1);

  printf("%d %d %d %d %d %d\n",
         x, y, z[1], z[2], p1, p2 );
  printf("%d %d %d\n",
         *p1, *p2, **p2);

}
int f1(int *a, int b, int *c, int **d)
{
   int x=16, y=3;

   *a = x / y / 2;      /* ! integer division ! */

   *d = c+1;

   **d = *(*d-2);

   return *c + *a;
}
```

MEMORY

Name	Address	Content/Value
x	12	
y	16	
z[0]	20	
z[1]	24	
z[2]	28	
z[3]	32	
p1	36	
p2	40	
	100	
a	104	
b	108	
c	112	
d	116	
x	120	
y	124	
	128	

OUTPUT

3. (20 pt) Write a function `count_common(int A[], int B[], int nA, int nB);` which takes two 1D arrays of integers and their sizes as parameters, and then counts the number of common numbers in both arrays.

   Assume that the numbers in each array are **unique** and **sorted** from smallest to largest.

   For example, if we have

```
int A[6] = {1, 2, 3, 6, 7, 9};
int B[5] = {1, 3, 4, 6, 8};
int n;
```

   Then when we call your function as

```
n = count_common(A, B, 6, 5);
```

   it should return 3 because we have three common numbers (1, 3, 6) in both A and B.

   (give your answer in the next page)

```
int count_common(int A[], int B[], int nA, int nB)
{
  int i=0, j=0, count=0;
```

4. (20 pt) You are given an NxN 2D-array of integers (matrix) and a 1D-array of N integers (query). You are asked to write two functions to
   - compare the given 1D array (query) with each **row** (left-to-right →) of the matrix;
   - compare the given 1D array (query) with each **column** (top-to-bottom ↓) of the matrix.

   If there is a match, these functions will print the matched row numbers and column numbers.

   Here is an example showing how the functions that you will implement in the next page can be used in main( ).

```
/* suppose std C libraries are included here */
#define N 4      /* this number can be changed */

main()
{
  int matrix[N][N] =    {{10, 10, 30, 45},
                         {14, 10, 32, 11},
                         {20, 30, 40, 50},
                         {35 ,45, 25, 15} };

  int query[N];
  int i;

  printf("Enter %d integers to search in 2D array ", N);
  for(i=0; i < N; i++){
    printf("Enter query[%d] = ", i);
    scanf("%d", &query[i]);
  }

  compare_with_each_row( matrix, query, N );
  compare_with_each_column( matrix, query, N );
}

/*
When the above program is executed, suppose a user enters
10 10 30 45 as the query.

Then the program should print:

Row 0 is the same as query
Column 1 is the same as query

*/
```

```
void compare_with_each_row(int m[][N], int q[], int n) // (left-to-right →)
{
```

```
void compare_with_each_column(int m[][N], int q[], int n) // (top-to-bottom ↓)
{
```

5. (20pt) Suppose you run the following program and enter **3** when prompted for the number of rectangles. Assume that computer has enough memory to meet all memory allocation requests. Now you are asked to first conceptually draw the memory snapshot or layout (i.e., a **diagram** similar to the one given in Class Assignment 2 (payroll), and implement a function to get the location (x, y), width and height of all rectangles from the user.

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

typedef struct point {
   int x;
   int y;
} pointT;

typedef struct rect {
     pointT loc;   /* loc: location */
     int w, h;     /* w: width, h: height  */
} rectT;

main()
{
  rectT   **R;
  int  num_r, i, j;
  printf("Enter the number of rectangles :");
  scanf("%d", &num_r);
  R = (rectT **) malloc(num_r * sizeof(rectT *));
  if (R==NULL) exit(-1);
  for(i=0; i < num_r; i++) {
    R[i] = (rectT *) malloc(sizeof(rectT));
    if (R[i] ==NULL) exit(-1);
  }

  Get_loc_w_h_Rectangles( R, num_r ); /* implement this */

}
```

Name:.....

a. (10pt) Conceptually draw the memory snapshot (i.e., a **diagram** similar to the one in Assignment 2). Remember the user enters **3** when prompted for the number of rectangles.

b. (10pt) Give the implementation of G**et_loc_w_h_Rectangles(R, num_r);** which simply asks the user to enter location(x, y), width, and height for each rectangle.

```
void Get_loc_w_h_Rectangles( rectT **R, int num_r )
{
```