CS 2123-001 Data Structures

Spring 2016 – Midterm1 -- Feb 11, 2016
You have 75 min. Good luck.
You can use only the 2-page C reference card posted in the class web page.

Name:…… Score: ……./100

Background Survey (5pt bonus credit)

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

Programming courses	TAKEN AT UTSA? If **YES**, give the instructor's name and term (e.g., Clark, Tosun, Robinson, Sherette or ... in Fall 2015).	If **NO**, and 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:		
CS 2123 Data Structures (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. Based on the review we did so far, do you think you have sufficient background to take CS 2123?

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

1. (20 pt) Implement a function `char *str_even_pos(char *s1);` which copies the characters from the even positions 0, 2, 4, ... of `s1` into a dynamically created new string and returns this new string.

   For example:
   After `s = str_even_pos("ABCDEFG");` `s` should be pointing to a new string `"ACEG"`.
   After `s = str_even_pos("ABCDEF");` `s` should be pointing to a new string `"ACE"`.

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

```
char *str_even_pos(char *s1) {
  /* 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=3, y=9, z[4]={0}, *p1, **p2;

  p1 = &z[1];

  p2 = &p1;

  *p1++ = 5;

  *++p1 = 8;

  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=5, y=14;

   *a = y % x / 3;

   *d = c-1;

   **d = *(*d+2) + **d;

   return *a + b;
}
```

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) Recall the **word search** game that you implemented in Assignment 2, where you searched the hidden words horizontally (left-to-right →), vertically (top-to-bottom ↓) or diagonally (left-top-to-right-bottom ↘). The words could be hidden in reverse directions too.

   Now you are asked to write a function that implements **reverse vertical** (bottom-to-top ↑) search to find out if a given word (a null terminated string) appears **reverse vertically** in the given grid (2D array of characters, where rows or columns are NOT null terminated). If the word appears reverse vertically, then your function should return 1 as well as the index values for the beginning row and column of the hidden word in the grid (br, bc); otherwise, returns 0.

   Suppose the following function is already implemented and it is available.

   ```
   int substrindex(char *str, char *substr);
   ```

   which returns -1 if `substr` is not in `str`; otherwise, it returns an index value showing where `substr` starts in `str`. This function expects both `str` and `substr` to be NULL terminated strings. If you want, you can use this function without implementing it.

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

```
/* suppose std C libs are included here */
#define ROW 4
#define COL 3      /* these numbers will be large in an actual program */

main()
{
  char g[ROW][COL] = {{'e','r','t'},
                      {'d','f','g'},
                      {'c','v','b'},
                      {'x','y','z'}};
  char w[128];
  int res, br, bc;

  printf("Enter the word you want to search in the grid : ");
  fgets(w, 127, stdin);

  res = reverse_vertical( g, w, &br, &bc);
  if(res==1)
     printf("Word %s appears reverse vertically at (%d, %d)\n", w, br, bc);
  else
     printf("Word %s is not found \n", w);
}
/*
For example when w is "vfr" the program should print
Word vfr appears reverse vertically at (2, 1)
*/
```

```
int reverse_vertical(char g[][COL], char *w, int *br, int *bc )
{
   char tmp[ROW+1];   /* this is a hint for you */
```

4. (20pt) Suppose you run the following program and enter **3** for all the integer values asked by the program. Assume computer has enough memory to meet all memory allocation requests. Now you are asked to conceptually draw the memory snapshot or layout (i.e., a **diagram** similar to the one that we gave in Assignment 3), and implement a function to free all the dynamically allocated memory.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct point {
   int x;
   int y;
} pointT;
typedef struct polygon {
     pointT *corners;
     int nc;          /* number of corners in the shape */
} polygonT;
main()
{
  polygonT   *p;
  int  num_p, i, j;
  printf("Enter the number of polygons :");
  scanf("%d", &num_p);
  p = (polygonT *) malloc(num_p * sizeof(polygonT));
  if (p==NULL) exit(-1);
  for(i=0; i < num_p; i++) {
    printf("Enter the number of corners of polygon %d :", i);
    scanf("%d", &p[i].nc);
    p[i].corners = (pointT *) malloc(p[i].nc * sizeof(pointT));
    if (p[i].corners==NULL) exit(-1);
    for(j=0; j < p[i].nc; j++){
      printf("Enter x y for corner %d in polygon %d:",j, i);
      scanf("%d %d", &p[i].corners[j].x, &p[i].corners[j].y);
    }
  }    /* suppose we did something with the polygons */
  FreePolygonsAndTheirCorners( p, num_p ); /* implement this */
}
```

Name:.....

a. (10pt) Conceptually draw the memory snapshot (i.e., a **diagram** similar to the one in Assignment 3). Remember the user enters 3 for all the integers in the program.

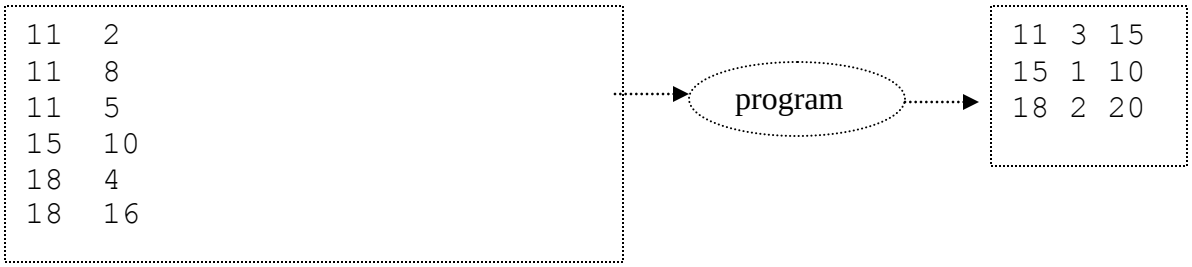
b. (10pt) Give the implementation of **`FreePolygonsAndTheirCorners,`** which frees all the dynamically allocated polygons and their corners.

```
void FreePolygonsAndTheirCorners(polygonT *p, int num_p ) {
```

5. (20pt) Suppose we have an employee data file (say emp.txt) containing the followings in each line: *employee ID* and *how many hours he/she worked in one day*. For the same employee, there might be more than one line if he/she works for more than one days. Luckily, the lines in the file are sorted with respect to employee ID. So all the work hours of an employee will appear back to back in the data file, as shown in the below example.

   Complete the following program that can read emp.txt file and print out the *employee ID*, *total number of days*, and *total number of hours* for each employee into an output file (say out.txt).

   For example, here is a sample **`emp.txt`** file with three employees and the expected **`out.txt`** file (an actual file may have more lines, so your program should be general enough to work with other files containing different number of employees or days):




```
#include <stdio.h>

int main(void)
{
  FILE *infp, *outfp; /* if needed, declare other variables here */
```

```
   if ((infp = fopen("emp.txt", "r"))==NULL){
      printf("Input file cannot be opened\n");
      return -1;
   }
   if ((outfp = fopen("out.txt", "w"))==NULL){
      printf("Output file cannot be opened\n");
      return -1;
   }
```