19CSE102 - Computer programming

Lab sheet -Pointers

Warm-up

1. Write a program to reverse an array using pointers.

2. Write a program to search an element in array using pointers.

Dynamic Memory allocation

3. Write a program with the following functions:
   i) double * Random(int n);
   Function dynamically allocates array of size n, where every element is a random real number between 0 and 1. Function returns pointer to this array.
   ii) double ** Matrix (double *A, int n, int r);
   Function dynamically allocates matrix with r rows and copies array A into newly created matrix (Note: number of columns has to be calculated from variables n and r. If A is smaller than matrix, fill blank space with zeros!). Function returns pointer to the matrix.
   iii) double * Max(double *A, int n);
   Function returns pointer to the maximum value in array A.

   Use all 3 functions in the main program: pass array created by Random function to the other 2 functions and print results.

   Measure execution time (in seconds) required by Random function to create and fill array and print it out.(Hint: Use clock() function in time.h headed file)

4. Write a program with a function that computes the scalar product between to arrays. (The scalar product of two arrays A and B with three elements is $A.B = A_xB_x + A_yB_y + A_zB_z$)
   In the main function, read a length from the user and create two arrays (at run time) of that length for which you calculate the scalar product.
   double scalar_product (int len , double * arr1 , double * arr2);

5. Without allocating memory dynamically, repeat the previous exercise using VLAs.

6. Dynamically allocate a 2D array and use it to create an n × 3 array filled with doubles of your choice. Interpreting each row in this array as the coordinate vector of a point in space, write a function that calculates the distance between two given points.

   (In three-dimensional space the Euclidean distance between (x_1, y_1, z_1) and (x_2, y_2, z_2) is

   $$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

7. Rewrite your code from Q5 and Q7 so that the two functions (scalar product and pair distance) are located in a separate C-file. Create a corresponding header file where you put the function prototypes. Recompile your program and make sure that everything works.

8. Given two unsorted arrays that represent (elements in every array are distinct), find the intersection of two arrays. For example, if the input arrays are:
   arr1[] = {7, 1, 5, 2, 3, 6}
   arr2[] = {3, 8, 6, 20, 7}
   Then your program should print Intersection as {3, 6, 7}. Note that the elements of intersection can be printed in any order. To find intersection do the following steps
   1) Initialize intersection I as empty.
   2) Find smaller array in size and sort the smaller array.
   3) For every element x of larger array, Binary Search x in smaller array. If x is present, then copy it to I.
   4) Return I.

   int* intersect(int* nums1, int nums1Size, int* nums2, int nums2Size, int* returnSize);

   (Note: The returned array must be malloced, assume caller calls free().)

9. Write the function threeColorsSort that takes as input an array of integers in the range of 0 and 2 (0, 1 and 2 only), and arranges them in an increasing order:

   void threeColorsSort(int * theArray, int arraySize)

   Then, write a program that asks the user for how many numbers to input, and then for the actual numbers. The program should then output the same numbers in ascending order. Make sure to free up your allocated memory.

Strings

10. Write a function that will take a string and return a count of each letter in the string. For example, "my dog ate my homework" contains 3 m's, 3 o's, 2 e's, 2 y's and one each of d, g, a, t, h, w, r and k. 2 Your function should take a single string argument and return a dynamically allocated array of 26 integers representing the count of each of the letters a . . z respectively. Your function should be case insensitive, i.e., count 'A' and 'a' as the occurrence of the letter a. [Hint: use the letter to integer conversion functions.] Do not count non-letter characters (i.e., spaces, punctuation, digits, etc.) Write a program that will solicit a string from the user using getline, call your letter frequency function and print out the frequency of each letter in the string. Do not list letters that do not occur at least once.

11. Implement an algorithm to alphabetically sort character strings using bubble sort. You have to do memory allocation dynamically. When comparing strings use strcmp from the library string.h. Write a program that asks the user for how many strings to input, what the maximum string length is and then the actual strings. The program should then output the same strings in alphabetical order (according to strcmp). The program should be able to handle an arbitrary number of strings of an arbitrary maximum length. Make sure to free up your allocated memory. The output has to be as follows:
    Number of strings: 5
    Maximum string length: 10

Give string 0: Hello
Give string 1: world!
Give string 5: big
Give string 6: number:
Give string 7: 1234567890
Input when sorted: 1234567890 Hello big number: world!

Perils of pointers

12. Does the following code run successfully to return 0 or does it generate a segmentation fault? If it runs fine, then what is the output? Otherwise explain why it segfaults.
(A segmentation fault occurs when a program attempts to access a memory location that it is not allowed to access.)

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

void populate(int *a) {
    int *parray = malloc(2 * sizeof(int));
    parray[0] = 37;
    parray[1] = 73;
    a = parray;
}
int main() {
    int *a = NULL;
    populate(a);
    printf("a[0] = %d and a[1] = %d\n", a[0], a[1]);
    return 0;
}
```

13. Write a basic program with pointers as directed below.
    a. Declare a pointer to an integer variable ptr.
       int * ptr;
    b. Use malloc to dynamically allocate memory for ptr.
       ptr = (int *) malloc(sizeof(int));
    c. Assign an integer value to the memory pointed to by ptr.
       *ptr = 10;
    d. Print the value pointed to by ptr to the terminal.
       printf("%d\n", *ptr);
    e. Free the ptr memory.
       free(ptr);

14. Perils of pointers: Variations of Q13 to simulate the problems due to mishandling of pointers.
    a. **A case of null pointer**: Access value without allocating the memory

i. Perform steps a and c (i.e. without b).
ii. Note down the error.

b. **Another case of null pointer**: Access value after freeing the memory
   i. Perform steps a, b, c, e and then d.
   ii. Note down the error.

c. **A case of memory leak**: Allocating memory without freeing.
   i. In an infinite loop, do steps a, b, c and d (i.e. without e).
   ii. Run the program for as long as it can.
   iii. The program + system will crash after some time. Restart your computer.

d. **A case of not allowing memory leak**: Allocating memory with freeing
   i. In an infinite loop, do steps a, b, c, d and e.
   ii. The program should run forever without crashing.
   iii. Press CTRL+C to stop execution.

e. **A case of lost pointer and memory leak**: Re-assigning a pointer to another location

```
int * p = (int *) malloc( sizeof(int) ); // p points to a location_1 in memory
*p = 5;
int * q = (int *) malloc( sizeof(int) ); // p points to a location_2 in memory
*q = 10;
p = q; // p is reassigned and both p and q point to location_2
```

   i. The access to location_1 is lost. It is impossible to retrieve the value 5.
   ii. It can't be freed either since the pointer is lost. This leads to memory leak.

f. **Another case of lost pointer**: Re-assigning a pointer to a new memory

```
int * p = (int *) malloc( sizeof(int) ); // p points to a location_1 in memory
*p = 5;
p = (int *) malloc( sizeof(int) ); // p reassigned to a new location (location_2)
*p = 10;
```

   i. The access to location_1 is lost. It is impossible to retrieve the value 5.
   ii. It can't be freed either since the pointer is lost. This leads to memory leak.

g. **A case of dangling pointer**: Freeing up a pointer when another is accessing the same location.

```
int * p = (int *) malloc( sizeof(int) ); // p points to a location_1 in memory
int * q = p; // Both p and q not point to location_1
free(p); // p frees location_1. The runtime system claims it.
*q = 10; // Can't access location_1 since it is no more program memory
```

   i. In short, q points to a location which does not legally belong to the program
   ii. Note down the error

h. **A case of messing up with pointer**: incorrect type casting leads

```
long * ptr = (long *) malloc( sizeof(int) ); // 4 bytes allocated
*ptr = 10; // This will access 8 bytes of memory
```

   i. Out of 8 bytes, only first 4 can be legally accessed.
   ii. The runtime system will report an error when trying to assign value 10.
   iii. Note down the error

****