Pointers

1.

void swap_any(void *p_var1, void *p_var2, size_t size);

implement the swap_any function. This particular function should know how to swap values of any type. Use the size parameter to determine the amount of bytes.

2.a

void arr_mix(int *arr, int n)

Implement the arr_mix function. The function will mix the elements. It will do so by:

1. Putting the lower half of the array in the even indexes of the array.
2. Putting the higher half of the array in the odd indexes of the array, reversed.
3. After doing so, move all elements one left in cycle.

10	20	30	40	50	60	70	80	90	100

10	100	20	90	30	80	40	70	50	60

100	20	90	30	80	40	70	50	60	10

2.b

void arr_mix_x(void *arr, int n, size_t size)

Do the same as a' only assume that the array's elements are the size you specify in size parameter.

Hint: you must duplicate the array to another buffer, you can use memcpy for that.

3.

Define two integers named xor_true and xor_false and initialize the variables to 1. Also define a pointer (type int *) named ptr.

The program will loop as long as the input is not zero. You are forbidden inputting directly to xor_true and xor_false (for example: you cannot write scanf("%d", &xon_true);).

Each time you receive input save it to xor_true / xor_false according to the last result of xor-ing all the bits in both variables together.

Print the variables' values in each iteration of the loop.

4.

Write a program which uses two integers named var1 and var2 and a pointer to int named pvar.

The program will loop repeatedly inputting each time a different variable unless the value received was negative. You are forbidden inputting directly to var1 and var2.

The program will end when both numbers are prime. It will then show the variables' name and value.

5.

Write a function which uses three integers named a, b, c.
The function should also define three pointers, pSmall, pMedium, pBig.

Input the three integers a, b and c using scanf, and save the addresses to the pointers accordingly.

From now on input data into pMedium using scanf. Each time pMedium is smaller than pSmall or pMedium is bigger than pBig switch the addresses.

The program ends when pBigger value minus pSmaller value > 100. Print which variable was bigger from which.

6.

Write a program which defines three variables:

1. An array of 10 integer named arr
2. A pointer to int named ptrArr
3. An integer named hops

Input during runtime the values of each element, so that values will not be bigger than 5, and init ptrArr so it will point to the first element.

Example of the array:

4	2	5	1	2	3	4	1	5	4

The higher half of the array (elements 6 to 10) will be multiplied by -1, and become negative.

Input into hops variable a positive number. From now on**, use only the pointer** and hop from each element to the next one using steps in the array. Print the absolute number each time you hop.

Example:

ptrArr hops ← 6

4	2	5	1	2	-3	-4	-1	-5	-4

4 → 2 → 4 → 5 → 1 → 4 → 5

7.a

Implement the following function:

char * gets_malloc()

The function will be responsible of inputting a string (initially be stored in a buffer big enough), and dynamically allocate the amount of bytes needed in the heap for the string.

When done the function returns the updated pointer received in the heap. If there is not enough memory return NULL.

7.b

Will the memory allocated need to be freed manually when done?

7.c

Define an array of strings that has 10 elements (char *strings[10];).

Try to sort the array (use any sorting method you know, the simplest is bubble sort)
hint: use strcmp() function.

8.

Define the next three variables:

Int len;
int *pArr;
int **ppTree;

Input the len variable, use it to dynamically allocate the amount of elements in pArr.

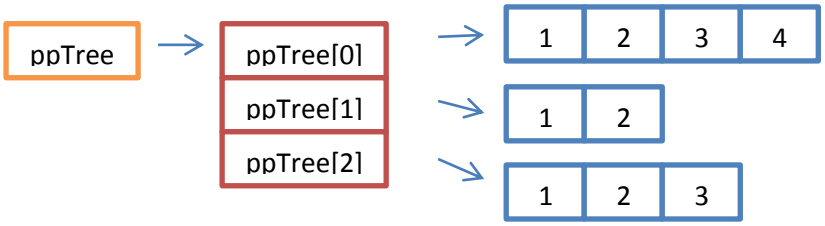
e.g len ←3 means the array pArr will have 3 elements.

Then input each element in the array.

Now use ppTree to create a dynamic array of pointers, with the length of len.

For each of those pointers dynamically allocate the amount of elements, using the corresponding element in pArr. Then initialize each of the endpoint element numbers from 1 to Length of each sub array.

e.g pArr[0] ← 4
pArr[1] ← 2
pArr[2] ← 3



9- Write a program which builds a dynamic array of bits.
Each element in the array differs from it's next by one bit.

The creation of the array is illustrated in the next paragraphs:

At first, the array will contain 2 elements and will be initialized as following:

Arr[0] <- 1
Arr[1] <- 0

1	0

Then, in each iteration the array's length will multiply and mirrored. The left half adds 1, the right half adds 0 as following:

1	0	0	1

11	10	00	01

Next iteration:

11	10	00	01	01	00	10	11

111	110	100	101	001	000	010	011

Next iteration:

111	110	100	101	001	000	010	011	011	010	000	001	101	100	110	111

1111	1110	1100	1101	1001	1000	1010	1011	0011	0010	0000	0001	0101	0100	0110	0111

* Hint – use function: *void * realloc (void *ptr, size_t size);* for resizing

a) The program will receive an input which indicates the amount of bits in each element of the array
b) Define the array in the following manner (your choice):
    a. Do it bitwise (each element is an integer of bits)

b. Do it as an array of bits (*harder*, each element is a pointer to another array of bits)

c) Write a function which prints the array, as presented in the previous illustration

d) Write a function which prints only the changes between each element

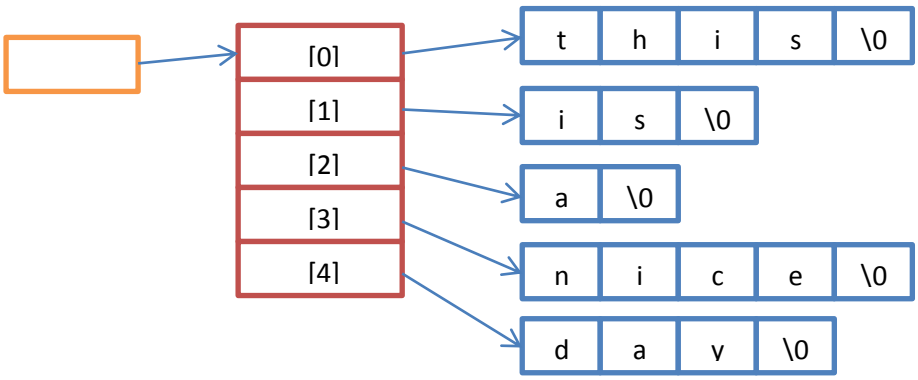
10- Write a program which implement the following function:

*char ** split(char *str, char dl);*

The way the result will be assembled is a pointer to strings, separated by the dl character.

For example:

Str = "this is a nice day", dl = ' '



11- A.

Create an array. The user will choose its length. The program will measure the amount of time for each sorting you implement.

Implement at least two sorting algorithms (more is optional), any of your choice, and give each a corresponding name.

for example quick_sort and bubble_sort.

- Mandatory, create a pointer to function, which receives the sorting functions. *void (* sort)(int *arr, int length);*.
- Mandatory, use clock function to measure the amount of clock cycles since the sorting started *clock_t clock();*.
- Optional, the array can be initialized randomly using the random or rand function *int random(int n);* / *int rand(void);*. Initialization for them is required. Recommended for large length arrays.

B.

Change the previous structure of the program, and create a struct, which represents a data type instead of int (for example - Point). The array will contain instances of the new struct you created.

implement your own comparison function for the struct you created. (For example, for Point - the distance from (0,0))

the previous sorting functions should change its integer comparison to your compare function. Implement the following:

```
struct <name> { … };   // your choice of a struct
int compare (void *elem1, void *elem2);    // compares the struct
void sort (int (* compare)(void *, void *), void *arr, int length);
```

- Mandatory, the sort function(s) should not be aware of the data type at no circumstances. Only the compare function is allowed to know and use the data type !

12- Encryption – part I :

Input a string(1) from user, and input a string key of chars.

the program should do XOR between the string and the key.

the following demonstrate the operation of the encryption:

String: "this string is an example of an unencrypted text"
Key: "my password"

String	t	h	i	s		s	t	r	i	n	g		i	s		a	n
Key	m	y		p	a	s	s	w	o	r	d	m	y		p	a	s
encrypt	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

- Mandatory, Implement the decryption as well.
- Optional(1), the string mentioned in the beginning of the question can also be the content of a file.

Encryption – part II :

Scramble the key, by Dividing It into two parts:

- If the key is even, add a character to it – '!' to make it odd.
- If the key is odd:
    - key1 = len(Key)/2
    - key2 = len(key)/2 +1

Use the key1 and key2 and do the same operation.
Demonstration:

Key: "my password" -> Key1 = "my pa", Key2 = "ssword"

String	t	h	i	s		s	t	r	i	n	g		i	s		a	n
Key1	m	y		p	a	m	y		p	a	m	y		p	a	m	y
Key2	s	s	w	o	r	d	s	s	w	o	r	d	s	s	w	o	r
encrypt	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#

- Optional, in case of equal chars between key1 and key2, use a single XOR. Recommended (because they undo each other).