Tutorial 5 : 30 August

1 Some clarifications

1.1 EOF and '\0'

EOF is a macro which expands to an integer constant expression with type int and an implementation dependent negative value but is very commonly -1. Note that the value -1 means it is not a character(char is from 0-255). You should understand it as just a signal given by system meaning end of data.

Question : Can I print the character EOF?
Answer : In short, no. EOF is not a character (in most modern operating systems). It is simply a state that applies to a file stream when the end of the stream is reached i.e. we cannot read anymore. But you can surely print the value of EOF (Which is generally -1 as mentioned above).

'\0' is a character with value 0. It is very commonly used as a string delimiter i.e string separator. Obviously, Computer needs to know then end of your sentence/paragraph or essay!. So, after any sequence of character data, we/system appends a '\0' character to keep a marker. This is very commonly used to find length of the string.

1.2 putchar()

putchar() is a standard function to print a character. It takes one argument, value of the character to be printed and prints it. As we can see in the following example, it behaves similar to printf("%c"...). In line 12, it prints the 'c' even when we try to give it 65 i.e. printing the corresponding character.

putchar() vs printf(): printf() is a generic printing function that works with multiple format specifiers and prints the corresponding string. putchar() just prints a character one at a time. You can verify that putchar() is better than printf() in terms of performance. But printf() wins in terms of flexibility and utility.

1.2.1 Problem 1

```
 1 #include <stdio.h>
 2 
 3 main( )
 4 {
 5   char c='A';
 6     printf ("%c%c%c\n", c, 'A', 65 );
 7     putchar (  c );
 8     putchar ( 'A' );
 9     putchar (  65 );
10     putchar ('\n');
11     return 0;
12 }
```

Output :
AAA
AAA

1.3 getchar()

getchar is a standard function for taking one character as input. getchar returns the character taken as input. Try, char c = getchar(); In case there is no more input to read, it returns EOF.

2 Find the output for all the following problems

2.1 Problem 1

```
1 #include <stdio.h>
2
3 main( )
4 {
5   char s[] = "I Love ESC101" ;
6   printf ( "%s\n", s );
7   printf ( "%c\n", s[2]);
8 }
```

Output :

```
I Love ESC101
L
```

Explanation :

- Note that in statement 5, you have declared an array without providing the size. Therefore the size automatically gets calculated from the string which the array is initialized with. For example, in this case an array if size 14 is declared with the following contents :

‘I’	‘ ’	‘L’	‘o’	‘v’	‘e’	‘ ’	‘E’	’S’	‘C’	‘1’	‘0’	‘1’	‘\0’

  Important point to note is that a ‘\0’ automatically gets appended at the end.

- Since there is a ‘\0’ appended at the end, therefore *printf()* is able to print the entire array with *%s*. Basically, the ‘\0’ acts like a marker for *printf* to indicate upto how far it has to print.

- Since internally strings are simply stored as arrays in C, thus you can access individual characters can simply be accessed using indices as done in statement 7 of the code.

Strings vs Char Array: Strings and character arrays, both contain sequences of characters which can be easily accessed. Let’s look at how they are different, a character array is similar to any other array adding easy access to individual items. While string also adds the fact that the sequence of characters can be seen a complete thing in itself. For example, we can print the whole string using printf("%s"...) instead of printing each character one by one. Source for more information.

2.2 Problem 2

```
 1 #include <stdio.h>
 2
 3 int main()
 4 {
 5     int i, j, k;
 6     char str[100] = "!setalocohc evol I";
 7     char str2[100];
 8
 9     for(i = 0; str[i] != '\0'; i++);
10     {
11        printf("Am I inside the loop?\n");
12     }
13
14     int length = i;
15     printf("length = %d\n", length);
16
17     for(j = 0, k = i-1; j < i; j++, k--)
18       str2[j] = str[k];
19     str2[j] = '\0';
20
21     printf("%s", str2);
22 }
```

Output :

```
Am I inside the loop?
length = 18
I love chocolates!
```

Explanation :

- The statement 11 is executed only once because it is not a part of body of for loop. Since there is a semicolon at the end of statement 9, the loop body is empty. As mentioned in previous tutorial sheet, if the braces are not specified, only the statement following immediately after the *for()* statement gets included in the body. So in this example, the body of loop contains an empty statement. Thus the loop in statement 9 is equivalent to

  ```
  for(i = 0; str[i] != '\0'; i++)
  {
      ;
  }
  ```

  Therefore at the end of the loop, the value of variable i can be used to compute the length of the string which is 18. Note that we generally exclude the '\0' character when we talk about the length. Also, loops like this is a common trick to find the length.

- In the second *for* loop, we simply read the *str* array from forward direction and store it *str2* from backward direction, thus reversing it. Again, statement 19 is not part of the for loop but statement 18 is. Important point to note is that we have to manually store a '\0' character at the end of the *str2* **if** we want to treat the array as a string (as done here in statement 21).

  Through this example, you can get some intuition about how to reverse a string.

2.3 Problem 3 : Complete the for loop to get the expected output

For a given array *x*, your code should swap the elements such that the first half of the array becomes the second half and vice-versa.
Consider the following code :

```c
#include <stdio.h>

int main() {
  int x[] = {0,1,2,3,4,5,6,7,8,9};
  int length = 10;

  ..        // To be filled by you

  for (  ..  ;  ..  ;  ..  ) {  // To be filled by you

  .. // To be filled by you

  }

  for (i = 0; i < length; i++) {
    printf("%d ", x[i]);
  }
}
```

You are allowed to do multi-statement additions in the above code at the specified points to get the expected output, which is

Expected Output : 5 6 7 8 9 0 1 2 3 4

Solution :
There are multiple solutions possible for this question. One of the possible solutions is shown below.

```c
1  #include <stdio.h>
2
3  int main() {
4    int x[] = {0,1,2,3,4,5,6,7,8,9};
5    int length = 10;
6
7    int temp, i, j;
8    for (i = 0, j = length/2; i<length/2; i++, j++) {
9      temp = x[i];
10     x[i] = x[j];
11     x[j] = temp;
12   }
13
14   for (i = 0; i < length; i++) {
15     printf("%d ", x[i]);
16   }
17 }
```

An important concept demonstrated in statements 9-11 is how to swap the value of two variables (here, x[i] and x[j]) using a temporary variable.

To try : Swap the value of two variables without using any extra variable.

3 Extra Problem

3.1 Reverse Order of Words in Array

For a given string *str*, your code should create a new array which contains the words of the string in reverse order. Assume that words are separated by spaces.

Input : I am a very good boy/girl
Output : boy/girl good very a am I

The outline of the algorithm is as follows (Note that this is not the actual code, these are hints meant to be useful in solving the problem):

```
str = Character Array (Consider "I am a very good boy/girl")
arr = New Character Array (We store the new string in this)

\\ Get the length of the string
l = length(string)

\\ Start from the right most point in the string
\\ Consider the right most word, find the beginning position of that word

Do it yourself;

\\ "I am a very good boy\girl"
\\ "I am a very good boy\girl_" , _ refers to our position
\\ "I am a very good _boy\girl" , now we are at the beginning of boy
\\ Now go over the word starting from the beginning index till the end index of the word
\\ "I am a very good b_oy\girl" , Step 1
\\ "b_            "
\\ Keep adding the word in another array while going over it
\\ "I am a very good bo_y\girl" , Step 2
\\ "bo_           "
\\ Continue till you reach the end of the word
\\ "I am a very good boy\girl_" , Step 3
\\ "boy\girl_                 "

Think of the rest yourselves;

\\ Repeat till you reach the start of the original array
```

Bonus Problem: The method described/hinted above uses another array to compute the reverse. Can you think of an algorithm to do it 'in-place'? In-place means that you will not use another intermediate array. You should modify the original array such that it stores the words in reverse order.