1. How many ways to store a string "Hello World!" in C? Where are they stored? Lists all the different memory storage ways in which this string can be stored

    a. There are several ways to store a string “Hello World!” in C. Some of given below:

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

// Global String Declarations
// 1.Global Character Array (Stored in Data Section)
char str8[] = "Hello World!";

// 2.Global Constant Character Array (Stored in Read-Only Data Section)
const char str9[] = "Hello World!";

// 3.Global Static Character Array (Stored in Data Section)
static char str10[] = "Hello World!";

int main() {
    // 4. String Literal (Stored in Read-Only Data Section)
    char *str1 = "Hello World!";


    // 5. Character Array (Stored in Stack)
    char str2[] = "Hello World!";


    // 6. Explicitly Initialized Character Array (Stored in Stack)
    char str3[13] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!', '\0'};


    // 7. Dynamically Allocated String (Stored in Heap)
    char *str4 = (char *)malloc(13);
    strcpy(str4, "Hello World!");


    // 8. Constant Character Array (Stored in Read-Only Data Section)
    const char str5[] = "Hello World!";

```

```c

    // 9. Static Character Array (Stored in Data Section)
    static char str6[] = "Hello World!";
;

    // 10. Static Pointer to String Literal (Stored in Read-Only Data Section)
    static char *str7 = "Hello World!";


    // 11. Volatile Character Array (Stored in Stack)
    volatile char str11[] = "Hello World!";

    // 12. Using strdup() (Stored in Heap)
    char *str12 = strdup("Hello World!");

    return 0;
}

```

2. What is the difference between strcpy and memcpy? Illustration
    a. Difference between **strcpy** and **memcpy:**

Feature	**strcpy**	**memcpy**
Purpose	Copies strings (null-terminated)	Copies raw memory (blocks of bytes)
Terminator Handling	Stops at null terminator ('\0')	Copies a fixed number of bytes, no null terminator
Use Case	Strings only	Any kind of memory (strings, structs, etc.)
Input Parameters	Source must be a string (null-terminated)	Source and destination can be any type of data
Terminator Copying	Includes null terminator ('\0')	Does not include null terminator
Example Use Case	Copying strings from one to another	Copying blocks of data (e.g., structs, arrays)

b. Illustration:

```c
#include <stdio.h>
#include <string.h>

int main() {
    // Example for strcpy
    char source_str[] = "Hello, World!";  // A string (null-terminated)
    char dest_str[20];                    // Destination array with enough space

    strcpy(dest_str, source_str);  // Copy string (including null terminator)
    printf("\n\nstrcpy result: %s\n", dest_str);  // Output: Hello, World!

    // Example for memcpy
    char source_data[] = { 'H', 'e', 'l', 'l', 'o', 0x01, 0x02 }; // raw data
    char dest_data[10];  // Destination array

    memcpy(dest_data, source_data, sizeof(source_data)); // Copy raw data (no null terminator)

    printf("memcpy result: ");
    for (int i = 0; i < sizeof(source_data); i++) {
        printf("0x%02x ", dest_data[i]);  // Output: 0x48 0x65 0x6c 0x6c 0x6f 0x01 0x02
    }
    printf("\n");

    return 0;
}

```

c. Output:

```
strcpy result: Hello, World!
memcpy result: 0x48 0x65 0x6c 0x6c 0x6f 0x01 0x02
PS C:\Users\Admin\Documents\bdcom_coding_zone\string>
```

3. Pointers-On-C.pdf 9.14 question 3:
   Write a function called **my_strcpy** that is similar to **strcpy** but will not overflow the destination array. The result of the copy must be a true string.

   a.

```c
#include <stdio.h>

void my_strcpy(char *dest, size_t dest_size, const char *src) {
    if (dest == NULL || src == NULL || dest_size == 0) {
        return; // Avoid null pointer issues or empty destination
    }

    size_t i;
    for (i = 0; i < dest_size - 1 && src[i] != '\0'; i++) {
        dest[i] = src[i];
    }

    dest[i] = '\0'; // Ensure null termination
}

int main() {
    char dest[6];  // Small buffer to test safety
    const char *src = "Hello World!"; // Longer than dest size

    my_strcpy(dest, sizeof(dest), src);

    printf("\n\nCopied String: \"%s\"\n", dest); // Expected: "Hello"

    return 0;
}
```

   b. Output:

```
Copied String: "Hello"
PS C:\Users\Admin\Documents\bdcom_coding_zone\string>
```

4. Pointers-On-C.pdf 9.14 question 5 :
   Write the function

   void my_strncat(char *dest, char *src, int dest_len);

   which concatenates the string in **src** to the end of the string in **dest**, making sure not to overflow the dest array, which is **dest_len** bytes long. Unlike **strncat**, this function takes into account the length of the string already in dest thereby insuring that the array bounds are not exceeded.

   a.

```c
#include <stdio.h>

void my_strncat(char *dest, char *src, int dest_len) {
    if (dest == NULL || src == NULL || dest_len <= 0) {
        return; // Prevent invalid input
    }

    int dest_current_len = 0;

    // Find the length of dest (but within bounds)
    while ( (dest_current_len < dest_len - 1) && (dest[dest_current_len] != '\0')) {
        dest_current_len++;
    }

    // Start appending src at the end of dest
    int i = 0;
    while (dest_current_len < dest_len - 1 && src[i] != '\0') {
        dest[dest_current_len] = src[i];
        dest_current_len++;
        i++;
    }

    // Null terminate the string
    dest[dest_current_len] = '\0';
}

int main() {
    char dest[12] = "Hello";  // 12 bytes allocated, "Hello" takes 6 (5 chars + '\0')
    char src[] = " World!!!";

    my_strncat(dest, src, sizeof(dest));

    printf("\n\nConcatenated String: \"%s\"\n", dest); // Expected: "Hello World"

    return 0;
}

```

b. output:

```
Concatenated String: "Hello World"
PS C:\Users\Admin\Documents\bdcom_coding_zone\string>
```

5. Find the average. There are 8 people in the group, and the current C language test scores are 210, 327, 413, 57145, 9154, 163, 23172, and 4081. The current device is an 8-bit-based PC, the maximum size is 8 bits. Only uint8 is available. You are allowed to use arrays, structs, or other simple variables. Please provide the average score of these 8 people.

a. C code of this problem:

```c
#include <stdio.h>
#include <stdint.h>

#define N 8  // Number of people


// Define a struct to hold a 32-bit number using four 8-bit values (MSB -> LSB)
typedef struct {
    uint8_t bytes[4];  // 4 bytes for the 32-bit value (MSB -> LSB)
} UInt32;

// Function to add two UInt32 numbers (correct handling of overflow/carry)
void add_uint32(UInt32 *sum, UInt32 num) {
    uint8_t carry = 0, temp_carry;

    // Add each byte from MSB to LSB, considering the carry
    for (int i = 3; i >= 0; i--) {

        // sum->bytes[i] = (sum->bytes[i] + num.bytes[i]) + carry;
        // carry         = carry_generate_from [(sum->bytes[i] + num.bytes[i]) + carry]);

        uint8_t temp = sum->bytes[i] + num.bytes[i];

        temp_carry = (temp < sum->bytes[i]) || (temp < num.bytes[i]);
                                            // Carry occurs if the sum overflows

        uint8_t temp2 = temp + carry;

        temp_carry += (temp2 < temp) || (temp2 < carry);
                                            // Carry occurs if the sum overflows


```

```
34
35          // final
36          carry = temp_carry;
37          sum->bytes[i] = temp2;
38
39      }
40
41  }
42
43  // Function to divide a UInt32 number by 8 using shifts (without exceeding byte limits)
44  UInt32 divide_by_8(UInt32 sum, uint8_t *remainder) {
45      UInt32 result = sum;
46      // uint8_t remainder = 0;
47
48      // Perform the division from MSB to LSB
49      for(int j = 0; j < 3; ++j){
50          for (int i = 3; i>= 0; i--) {
51
52              if(i < 3 && (result.bytes[i]&1)){
53                  result.bytes[i+1] |=(1 << 7);
54                                      // set bit 1 on most significant bit of right hand bytes
55              }
56              if(i == 3){
57                  *remainder >>= 1;
58                  if(result.bytes[i]&1){
59                      *remainder |= (1 << 7);
60                                      //set bit 1 on most significant bit of remainder bytes
61                  }
62              }
63
64              result.bytes[i] >>= 1;
65          }
66      }
67
68      return result;
69
```

```
73
74  int main() {
75      // Define scores as 32-bit values using the struct (MSB -> LSB)
76      UInt32 scores[N] = {
77          {{0, 0, 0, 210}},   // 210
78          {{0, 0, 1, 71}},    // 327
79          {{0, 0, 1, 157}},   // 413
80          {{0, 0, 223, 57}},  // 57145
81          {{0, 0, 35, 194}},  // 9154
82          {{0, 0, 0, 163}},   // 163
83          {{0, 0, 90, 132}},  // 23172
84          {{0, 0, 15, 241}}   // 4081
85      };
86
87      // Initialize the total sum as zero
88      UInt32 total_sum = {{0, 0, 0, 0}};
89
90      // Sum all the scores
91      for (uint8_t i = 0; i < N; i++) {
92          add_uint32(&total_sum, scores[i]);
93
94      }
95
96      // Divide the total sum by 8
97      uint8_t remainder = 0;
98      UInt32 avg = divide_by_8(total_sum, &remainder);
99                                  // passing total sum and remainder address and returned avg
100
101     // Print the average score in hex format with remainder
102     printf("Average Score: 0x%x%x%x%x.%x\n",
103         avg.bytes[0], avg.bytes[1], avg.bytes[2], avg.bytes[3], remainder);
104
105
106     return 0;
107 }
108
```

b. output (in hex) :

```
Average Score: 0x002e39.20
PS C:\Users\Admin\Documents\bdcom_coding_zone\string>
```