1.Problem 1: Array Element Access
// Write a program in C that demonstrates the use of a pointer to a const array of integers. The
program should do the following:
// 1. Define an integer array with fixed values (e.g., {1, 2, 3, 4, 5}).
// 2. Create a pointer to this array that uses the const qualifier to ensure that the elements cannot be
modified through the pointer.
// 3. Implement a function printArray(const int *arr, int size) to print the elements of the array using
the const pointer.
// 4. Attempt to modify an element of the array through the pointer (this should produce a
compilation error, demonstrating the behavior of const).
// Requirements:
// a. Use a pointer of type const int* to access the array.
// b. The function should not modify the array elements.

```
#include <stdio.h>

void printArray(const int *arr, int size) {
   for (int i = 0; i < size; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}

int main() {
   int array[] = {1, 2, 3, 4, 5};
   const int *ptr = array;

   printArray(ptr, 5);

   return 0;
}
```

OUTPUT

1 2 3 4 5

2. Problem 2: Protecting a Value
// Write a program in C that demonstrates the use of a pointer to a const integer and a const pointer
to an integer. The program should:
// 1. Define an integer variable and initialize it with a value (e.g., int value = 10;).
// 2. Create a pointer to a const integer and demonstrate that the value cannot be modified through

the pointer.
// 3. Create a const pointer to the integer and demonstrate that the pointer itself cannot be changed
to point to another variable.
// 4. Print the value of the integer and the pointer address in each case.

```
#include <stdio.h>

int main() {
   int value = 10;
   const int *ptr_to_const = &value;
   int *const const_ptr = &value;

   printf("Value using ptr_to_const: %d\n", *ptr_to_const);
   printf("Address of ptr_to_const: %p\n", (void *)ptr_to_const);

   printf("Value using const_ptr: %d\n", *const_ptr);
   printf("Address of const_ptr: %p\n", (void *)const_ptr);

   *const_ptr = 20;
   printf("Modified value: %d\n", value);

   return 0;
}
```

OUTPUT

```
Value using ptr_to_const: 10
Address of ptr_to_const: 0x7ffee12abec0
Value using const_ptr: 10
Address of const_ptr: 0x7ffee12abec0
Modified value: 20
```

3. Problem: Universal Data Printer
// You are tasked with creating a universal data printing function in C that can handle different types
of data (int, float, and char*). The function should use void pointers to accept any type of data and
print it appropriately based on a provided type specifier.
// Specifications
// Implement a function print_data with the following signature:
// void print_data(void* data, char type);Parameters:
// data: A void* pointer that points to the data to be printed.
// type: A character indicating the type of data:
// 'i' for int
// 'f' for float
// 's' for char* (string)

// Behavior:
// If type is 'i', interpret data as a pointer to int and print the integer.
// If type is 'f', interpret data as a pointer to float and print the floating-point value.
// If type is 's', interpret data as a pointer to a char* and print the string.
// In the main function:
// Declare variables of types int, float, and char*.
// Call print_data with these variables using the appropriate type specifier.

```
#include <stdio.h>

void print_data(void *data, char type) {
   if (type == 'i') {
      printf("Integer: %d\n", *(int *)data);
   } else if (type == 'f') {
      printf("Float: %.2f\n", *(float *)data);
   } else if (type == 's') {
      printf("String: %s\n", (char *)data);
   } else {
      printf("Unknown type\n");
   }
}

int main() {
   int i = 42;
   float f = 3.14;
   char *s = "Hello, World!";

   print_data(&i, 'i');
   print_data(&f, 'f');
   print_data(s, 's');

   return 0;
}
```

OUTPUT

```
Integer: 42
Float: 3.14
String: Hello, World!
```

4. Requirements
// In this challenge, you are going to write a program that tests your understanding of char arrays
// • write a function to count the number of characters in a string (length)
// cannot use the strlen library function
// function should take a character array as a parameter
// should return an int (the length)

// • write a function to concatenate two character strings.cannot use the strcat library function
// • function should take 3 parameters
// char result()
// const char str1[]
// const char str2[]
// can return void
// • write a function that determines if two strings are equal
// cannot use strcmp library function
// • function should take two const char arrays as parameters and return a Boolean of true if they are
equal and false otherwise

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

int length(const char str[]) {
  int len = 0;
  while (str[len] != '\0') {
    len++;
  }
  return len;
}

void concatenate(char result[], const char str1[], const char str2[]) {
  int i = 0, j = 0;

  while (str1[i] != '\0') {
    result[i] = str1[i];
    i++;
  }

  while (str2[j] != '\0') {
    result[i] = str2[j];
    i++;
    j++;
  }

  result[i] = '\0';
}

bool is_equal(const char str1[], const char str2[]) {
  int i = 0;
  while (str1[i] != '\0' || str2[i] != '\0') {
    if (str1[i] != str2[i]) {
      return false;
    }
```

```
        i++;
    }
    return true;
}

int main() {
    char str1[] = "Hello";
    char str2[] = "World";
    char result[100];

    printf("Length of str1: %d\n", length(str1));
    printf("Length of str2: %d\n", length(str2));

    concatenate(result, str1, str2);
    printf("Concatenated string: %s\n", result);

    printf("Are str1 and str2 equal? %s\n", is_equal(str1, str2) ? "True" : "False");
    printf("Are str1 and \"Hello\" equal? %s\n", is_equal(str1, "Hello") ? "True" : "False");

    return 0;
}
```

OUTPUT

```
Length of str1: 5
Length of str2: 5
Concatenated string: HelloWorld
Are str1 and str2 equal? False
Are str1 and "Hello" equal? True
```

5. string length without using strlen

```
#include <stdio.h>

int main() {
    char str1[] = "Good";
    char str2[] = "Morning";
    int i, length1 = 0, length2 = 0;
    for (i = 0; str1[i] != '\0'; i++) {
        length1++;
    }
    for (i = 0; str2[i] != '\0'; i++) {
        length2++;
    }
    printf("Length of string 1: %d\n", length1);
    printf("Length of string 2: %d\n", length2);
```

```
    return 0;
}
```

OUTPUT

```
Length of string 1: 4
Length of string 2: 7
```