1.

```c
/*
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);
int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    int const *const ptr = arr;
    printArray(ptr, 5);
    //ptr[0] = 6; //error: assignment of read-only location
    return 0;
}

void printArray(const int *arr, int size) {
    //arr[0] = 1; //error: assignment of read-only location '*arr'
    for (int i = 0; i < size; i++) {
        printf("%d ", *(arr + i));
    }
```

```
    printf("\n");
}
```

```
PS C:\Users\betti\Desktop\Training\Day9> gcc -o task1 task1.c
task1.c: In function 'main':
task1.c:28:12: error: assignment of read-only location '*ptr'
   28 |     ptr[0] = 6; //error: assignment of read-only location
      |            ^
task1.c: In function 'printArray':
task1.c:33:12: error: assignment of read-only location '*arr'
   33 |     arr[0] = 1; //error: assignment of read-only location '*arr'
      |            ^
```

```
PS C:\Users\betti\Desktop\Training\Day9> ./task1
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.

Requirements:

    a. Use the type qualifiers const int* and int* const appropriately.

    b. Attempt to modify the value or the pointer in an invalid way to show how
the compiler enforces the constraints.
*/

#include <stdio.h>
int main() {
    int value1 = 10, value2 = 20;
```

```c
    int const *ptr1 = &value1;
    //*ptr1 = 30; //error: assignment of read-only location *ptr1'
    printf("Value: %d \nAddress: %p\n", value1, ptr1);
    int *const ptr2 = &value2;
    //ptr2 = &value1; //error: assignment of read-only variable ptr2'
    printf("Value: %d \nAddress: %p\n", value2, ptr2);
    return 0;
}
```

```
PS C:\Users\betti\Desktop\Training\Day9> gcc -o task2 task2.c
task2.c: In function 'main':
task2.c:25:11: error: assignment of read-only location '*ptr1'
   25 |     *ptr1 = 30; //error: assignment of read-only location *ptr1'
      |           ^
task2.c:28:10: error: assignment of read-only variable 'ptr2'
   28 |     ptr2 = &value1; //error: assignment of read-only variable ptr2'
      |          ^
```

```
PS C:\Users\betti\Desktop\Training\Day9> ./task2
Value: 10
Address: 000000D3D4DFFA7C
Value: 20
Address: 000000D3D4DFFA78
```

3.

```c
/*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.

Example output:
Input data: 42 (int), 3.14 (float), "Hello, world!" (string)
Output:
Integer: 42
Float: 3.14
String: Hello, world!

Constraints
1. Use void* to handle the input data.
2. Ensure that typecasting from void* to the correct type is performed within the
print_data function.
3. Print an error message if an unsupported type specifier is passed (e.g., 'x').
*/

#include <stdio.h>
#include <string.h>
void print_data(void* data, char type);
int main() {
   int int_type;
   float float_type;
   char str[100];

   printf("Enter the inputs(int, float,string)\n");
   scanf("%d %f %[^\n]", &int_type, &float_type, str);
   printf("Input data: %d (int), %0.2f (float), \"%s\" (string)\n", int_type,
float_type,str);

   printf("Output:\n");
   print_data(&int_type, 'i');
   print_data(&float_type, 'f');
   print_data(str,'s');
   return 0;
}
```

```c
void print_data(void* data, char type) {
    switch (type) {
        case 'i': {
            int* int_ptr = (int*)data;
            printf("Integer: %d\n", *int_ptr);
            break;
        }
        case 'f': {
            float* float_ptr = (float*)data;
            printf("Float: %.2f\n", *float_ptr);
            break;
        }
        case 's': {
            char* str_ptr = (char*)data;
            printf("String: %s\n", str_ptr);
            break;
        }
        default:
            printf("Error: Unsupported type specifier '%c'\n", type);
    }
}
```

```
PS C:\Users\betti\Desktop\Training\Day9> ./task3
Enter the inputs(int, float,string)
42 3.14 Hello, World!
Input data: 42 (int), 3.14 (float), "Hello, World!" (string)
Output:
Integer: 42
Float: 3.14
String: Hello, World!
```

4.

```c
/*Write a function to count the number of characters in a string(length)
    cannot use 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
Requiremets:
    cannot use strcat library function
    function should take 3 parameters and can return void
        char result[]
        const char str1[]
```

```
        const char str2[]

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 *a);
void concat(char result[], const char str1[], const char str2[]);
bool equal(const char *a, const char *b);
int main() {
    char result[100];
    char str1[20];
    char str2[20];
    printf("Enter the first string: ");
    scanf("%s", str1);
    printf("Enter the second string: ");
    scanf("%s", str2);

    char op;
    printf("\nl: to find length of the strings\nc: to concatenate the strings\nm:
to compare the strings");
    printf("\nChoose the operation: ");
    scanf(" %c", &op);
    printf("\n");
    switch (op)
    {
    case 'l':
        int len1 = length(str1);
        printf("Length of first string: %d\n", len1);
        int len2 = length(str2);
        printf("Length of second string: %d\n", len2);
        break;

    case 'c':
        concat(result, str1, str2);
        break;

    case'm':
        bool result = equal(str1, str2);
```

```
        if (result) {
            printf("The strings are equal.\n");
        } else {
            printf("The strings are not equal.\n");
        }
        break;

    default:
        printf("Invalid operation!\n");
        break;
    }

    return 0;
}

int length(const char *a) {
    int count = 0;
    while (a[count]!= '\0') {
        count++;
    }
    return count;
}

void concat(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';
    printf ("The result of concatenation: %s\n", result);
}

bool equal(const char *a, const char *b) {
    while (*a!= '\0' && *b!= '\0') {
        if (*a!= *b) {
            return false;
        }
```

```
        a++;
        b++;
    }
    if (*a!= '\0' || *b!= '\0') {
        return false;
    }
    return true;
}
```

```
PS C:\Users\betti\Desktop\Training\Day9> ./task4
Enter the first string: hello
Enter the second string: world

l: to find length of the strings
c: to concatenate the strings
m: to compare the strings
Choose the operation: l

Length of first string: 5
Length of second string: 5
PS C:\Users\betti\Desktop\Training\Day9>  ./task4
Enter the first string: hello
Enter the second string: world

l: to find length of the strings
c: to concatenate the strings
m: to compare the strings
Choose the operation: c

The result of concatenation: helloworld
PS C:\Users\betti\Desktop\Training\Day9> ./task4
Enter the first string: hello
Enter the second string: world

l: to find length of the strings
c: to concatenate the strings
m: to compare the strings
Choose the operation: m

The strings are not equal.
```