// 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>
int printArray(const int *arr,int size);
int main()
{
    int arr[5]={1,2,3,4,5};
    const int *p=arr;
    printf("Array elements:");
    printArray(p,5);
    //*p=7;   //compilatin error
    arr[0]=10;
    printf("Modified array elements are:");
    printArray(p,5);
    return 0;
}
int printArray(const int *arr,int size)
{
    for(int i=0;i<size;i++)
    {
        printf("%d",*(arr+i));
    }
    printf("\n");

}
```

Output:

```
Array elements:12345
Modified array elements are:102345
```

// 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.
// has context menu
#include<stdio.h>
int main()
{
   int value=10;
   int value_b=20;
   const int *p=&value;
   //*p=30;
   printf("value=%d,address of value=%p\n",*p,p);
   p=&value_b;
   printf("value=%d,address of value=%p\n",*p,p);
   int *const val=&value;
   *val=30;
   printf("value=%d,address of value=%p\n",*val,val);
   //val=&value_b  //compilation erorr
   return 0;
}
```

Output:

```
value=10,address of value=0x7ffc772f997c
value=20,address of value=0x7ffc772f9978
value=30,address of value=0x7ffc772f997c
```

```
#include<stdio.h>
int main()
{
   char str1[]="hello";
   char str2[]="world";
   int len1=0,len2=0;
   for(int i=0;str1[i]!='\0';i++)
   {
      len1++;
   }
   for(int i=0;str2[i]!='\0';i++)
   {
      len2++;
   }
   printf("Length of the first string: %d\n", len1);
   printf("Length of the second string: %d\n", len2);

}
```

Output:

```
Length of the first string: 5
Length of the second string: 5
```

// 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>
void print_data(void* data,char type);
int main()
{
    int int_type;
    float float_type;
    char string[100];
    printf("Enter the integer:");
    scanf("%d",&int_type);
    printf("Enter the float:");
    scanf("%f",&float_type);
    printf("Enter the string:");
    scanf("%s",&string);
    print_data(&int_type,'i');
    print_data(&float_type,'f');
    print_data(string,'s');
    return 0;
}
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("Error,unsupported type specifier '%c' is passed \n",type);
    }
}
```

Output:

```
Enter the integer:42
Enter the float:3.14
Enter the string:Hello,World!
Integer:42
Float:3.14
String:Hello,World!
```

```
// 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>
int str_len(const char str[]);
char str_concat(char res[], const char str1[], const char str2[]);
int str_equal(const char str1[], const char str2[]);
int main()
{
    const char str1[]="Hello";
    const char str2[]="World";
    char res[20];
    int len=str_len(str1);
    printf("The length of string is:%d\n",len);
    str_concat(res,str1,str2);
    printf("Concatenated string:%s\n",res);
    if(str_equal(str1,str2))
    {
        printf("str1 and str2 are equal.\n");
    }
    else
    {
        printf("str1 and str2 are not equal.\n");
    }
    return 0;
}
int str_len(const char str[])
{
    int len=0;
    while(str[len]!='\0')
    {
        len++;
    }
    return len;
}
char str_concat(char res[],const char str1[],const char str2[])
{
    int i,j=0;
    while(str1[i]!='\0')
    {
        res[i]=str1[i];
        i++;
    }
    while(str2[j]!='\0')
    {
        res[i]=str2[j];
        i++;
        j++;
    }
    res[i]='\0';
}
int str_equal(const char str1[],const char str2[])
{
    int i=0;
    while(str1[i]!='\0' || str2[i]!='\0')
    {
        if(str1[i]!=str2[i])
        {
            return 0;
        }
        i++;
    }
    return 1;
}
```

Output:

```
The length of string is:5
Concatenated string:HelloWorld
str1 and str2 are not equal.
```