```
1.
#include <stdio.h>

int main(){

    int a[] ={1,2,3};

    printf("Address of A[0] = %p\n",a);

    printf("001the element at the 0th index = %d \n",a[0]);

    printf("002the element at the 0th index = %d \n",*(a+0));

    printf("Address of A[1] = %p\n",a+1);

    printf("001the element at the 1st index = %d \n",a[1]);

    printf("002the element at the 1st index = %d \n",*(a+1));

    int *ptr = &a[0];

}
```

2.

```
// example pointer arithmetic


#include <stdio.h>
int main(){
    int a[] = {1,2,3,4,5,6,7,8,9};
    int *ptr = a;//initializing the pointer with address of array a[]
    printf("\nAddress of a[1] = %p\n\n",ptr+1);
    ptr = &a[1];//reinitializing the pointer to the ellement present in the
1st index
    printf("\nAddress of a[1] = %p\n\n",ptr);
    for(int i=0;i<9;i++){
        printf("a[%d] = %d ->",i,*(ptr+i));
    }
    printf("\n\n");
    *(ptr+3)=8;//element in the third index got changed from 4 to 8
    for(int i=0;i<9;i++){
        printf("a[%d] = %d ->",i,*(ptr+i));
    }
    return 0 ;
}
```

3.

```
//Passing an array as a parameter of an array.
//n represent the number of elements in the array
#include <stdio.h>
int addArray(int array[],int n);
int main(){
    int a[10]= {0,1,2,3,4,5,6,7,8,9};
    int sum = 0;
    sum = addArray(a,10);
    printf("\nSum = %d",sum);
    return 0 ;
}
int addArray(int array[],int n){
    int arSum = 0;
    for(int i = 0;i<n;i++){
        arSum+=array[i];
    }
    return arSum;
}
```

4.

```
//TASK : To use concept of pointer and Passing a pointer that points to an
array as a parameter of the fn.
#include <stdio.h>
int addArray(int *p,int n);
int main(){
    int a[10]= {0,1,2,3,4,5,6,7,8,9};
    int *ptr = a;
    int sum = 0;
    sum = addArray(ptr,10);
    printf("\nSum = %d",sum);
    return 0 ;
}
int addArray(int *pt,int n){
    int arSum = 0;
    for(int i = 0;i<n;i++){
        arSum+=(*pt+i);
    }
    return arSum;
}
```

5.

```
/*
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[] = {1,2,3,4,5};
    int const *ptr = arr;
    printarray(ptr,5);
    return 0;

}
void printarray(const int *arr,int size)
{
    for(int i=0;i<size;i++){

        printf("%d-->%d",i,*arr+i);
    }
    //*arr = 10;

    /*//by initializing this it will produce an error like this  error:
assignment of read-only location '*arr'
            //*arr = 10; */

}
```

6.

```c
/*
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 value = 10;

    const int *ptr = &value;//Created a pointer to a const integer
    //We are going to demonstarte that the value cannot be changed using the
pointer.
    printf("\nValue = %d,Address = %p\n",*ptr,ptr);
    /*
    *ptr = 50;
            ->error: assignment of read-only location '*ptr'

    */
    //int anotherValue = 20;
    int *const constPtr = &value;

    printf("\nConst pointer to int:\n");
    printf("Value: %d, Address: %p\n", *constPtr, constPtr);
    *constPtr = 15; // This is allowed because the value pointed to can be
modified
    printf("Modified Value: %d\n", *constPtr);
    //constPtr = &anotherValue; // Uncommenting this line will cause a
compilation error [error: assignmen of read-only variable 'constPtr']
    return 0;
}
```

7.

```c
#include <stdio.h>
int main(){
    char name[] = {'r','o','y'};
    printf("Size of name = %d",sizeof(name));
    return 0 ;
}
```

8.

```c
#include <stdio.h>

int stringSize(const char *str) {
    int count = 0;
    while (str[count] != '\0') { // Iterate until the null terminator
        count++;
    }
    return count; // Return the count of characters
}

int main() {
    char str[] = "Hello, World!";

    // Call the function to calculate the size
    int s = stringSize(str);

    // Print the result
    printf("The size of the string \"%s\" is: %d\n", str, s);

    return 0;
}
```

9.

```c
//when press l : find length of the string,c : concatinate
#include <stdio.h>
#include <string.h>
int main(){
    char name[] ="Abhinav";
    printf("The length of the name is = %d",strlen(name));
    return 0;
}
```

10.

```
#include <stdio.h>
#include <string.h>
int main(){
    char name[] ="Abhinav";
    char Initials[10];
    printf("The length of the name is = %d\n",strlen(name));
    strcpy(Initials,name);
    printf("initials = %s",Initials);
    return 0;
}
```

11.

```
/*Qn. 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_val = 42;
    float float_val = 3.14f;
    char* str_val = "Hello, world!";
    print_data(&int_val, 'i');
    print_data(&float_val, 'f');
    print_data(&str_val, 's');
    print_data(&int_val, 'x');
    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'\n", type);
    }
}
```

12.

```
/*Qn. write a function to concatenate two character strings
• cannot use the strcat library function
• function should take 3 parameters
• char result
• const char str10
•const char str2[]
•can return void*/
```

```c
#include <stdio.h>
void concatenate(char result[], const char str1[], const char str2[]);
int main() {
    const char str1[] = "Bhavana";
    const char str2[] = "Baiju";
    char result[100];
    concatenate(result, str1, str2);
    printf("The result of concatenation is: %s\n", result);
    return 0;
}
void concatenate(char result[], const char str1[], const char str2[])
{

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

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

    result[i] = '\0';
}
```

13.

```c
/*
Qn.• 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*/

int main()
int my_strcmp( char *str1, char *str2);
{

 t char str1[]="Hello";
 t char str2[]="Hello";

 int res = my_strcmp(str1,str2);
 if(res == 0)
 {
 printf("Not equal\n");
 }
 else
 {
 printf("Equal\n");
 }
 return 0;

}
int my_strcmp( char *str1, char *str2)
{

 int i = 0;
 while (str1[i] != '\0' && str2[i] != '\0')
 {
 if (str1[i] != str2[i]) {
 return 0; // Strings are not equal
 }
 i++;
 }
 return 1;
}
```