```
/*
int const* ===>value becomes constant but the pointer is modifiable
int *const  ===>value become modifiable but the pointer becomes constant
int const * const ===> both are unalterable
*/


#include <stdio.h>

int main()
{
   int num = 800;
   printf("001num = %d \n",num);
   int const *const pNum = &num;
   printf("001pNum = %p \n",pNum);

   int num1 = 900;
   pNum = &num1;

   return 0;
}
```

Void in Place of datatypes(usage of void pointers)

```
#include<stdio.h>

int main(){

   int i=1234;

   float pi=3.14;

   char c='A';

   void *ptr;

   ptr=&i;

   printf("i=%d \n",*(int *)ptr);

   ptr=&pi;

   printf("pi=%f \n",*(float*)ptr);

   ptr=&c;

   printf("c=%c \n",*(char*)ptr);
```

```
   return 0;

}
```

Using a[i] and *(a+i)

```
#include<stdio.h>

int main(){
   int a[]={1,2,3};
   // int *ptr=a;//without using &
   // int *ptr=&a[0];//with using &,when we use & we need to gave the index
position also;
   printf("Address of A[0] =%p\n",a); //starting address of a array ;
   printf("Address of A[1] =%d\n",a[1]); //element at index 1  ;
   printf("Address of A[1] =%d\n",*(a+1)); //element at index 1  ;
   printf("Address of A[1] =%d\n",*(a)); //element at index 0  ;
   printf("Address of A[2] =%p\n",a+2); //address of a[2];
   // printf("ptr = %p",ptr);//gives address of first element of array stored inside a
pointer;

}
```

Modifying third element in an array using pointers.

```
#include<stdio.h>

int main(){
   int a[]={1,2,3,4,5,6,7,8,9};
   int *ptr=a;//Initialize the pointer with address of array a[]
   for(int i=0;i<9;i++){
      printf("a[%d]=%d-->",i,*(ptr+i));
   }
   printf("\n");
   *(ptr+3)=8;

   for(int i=0;i<9;i++){
      printf("a[%d]=%d-->",i,*(ptr+i));
   }
}
```

Example

```
#include<stdio.h>

int main(){
   int a[]={1,2,3,4,5,6,7,8,9};
   printf("Address of a[1]=%p\n",a+1);
   int *ptr=a;//Initialize the pointer with address of array a[]
   printf("Address of a[1]=%p\n",ptr+1);
```

```
//reinitialize the pointer to the element present in the 1st index;
ptr=&a[1];
printf("Address of a[1]=%p\n",ptr);
printf("Address of a[2]=%p\n",ptr+1);

}
```

Without using pointers

```
//n represents number ofelements 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("Sum =%d\n",sum);

   return 0;

}
int addArray(int array[],int n){
   int arSum=0;
   for(int i=0;i<n;i++){
      arSum=arSum+array[i];
   }
   return arSum;
}
```

By using pointers

```
#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("Sum = %d\n", sum);

   return 0;
}

int addArray(int *array, int n) {
```

```
    int arSum = 0;
    for (int i = 0; i < n; i++) {
       arSum += *(array + i); // Using pointer arithmetic to access array elements
    }
    return arSum;
}
```

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 numbers[]={1,2,3,4,5};

   const int *ptr=numbers;

   printArray(ptr,5);

   // ptr[2]=10;

   // printf("ptr[2]=%p",ptr);

}

void printArray(const int *arr,int size){
```

```
    for(int i=0;i<size;i++){
        printf("Element at numbers[%d] : %d\n",i,arr[i]);
    }
}
```

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;

   // *ptr=20;
   printf("value =%d,address=%p \n",value,ptr);

   int num1=20;
   int *const constptr=&value;
   // constptr=&num1;
   printf("value %d,address=%p \n",value,ptr);

}
```

Strings

```
#include <stdio.h>

int main() {

   char str1[] = "To be or not to be!";
   char str2[] = "that is the question";

   int len1=0;
   while (str1[len1] != '\0') {
      len1++;
   }
    printf("Length of '%s' is: %d\n", str1, len1);
   len1=0;
   while (str2[len1] != '\0') {
      len1++;
   }


   printf("Length of '%s' is: %d\n", str2, len1);

   return 0;
}
```

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 itype = 42;
   float ftype = 3.14;
   char stype[] = "Hello, world!";

   print_data(&itype, 'i');
   print_data(&ftype, 'f');
   print_data(stype, 's');

   return 0;
}
```

```
void print_data(void * data, char type) {
   switch (type) {
      case 'i':
         printf("Integer: %d\n", *(int*)data);
         break;
      case 'f':
         printf("Float: %.2f\n", *(float*)data);
         break;
      case 's':
         printf("String: %s\n", (char*)data);
         break;
   }
}
```

- 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>

void concat(char *str1, char *str2, char *res);
int countCharacters(const char *str);
int areStringsEqual(const char *str1, const char *str2);

int main() {
   char string1[20];
   char string2[20];
   char result[40];

   printf("Enter the first string: ");
   scanf("%s", string1);

   printf("Enter the second string: ");
   scanf("%s", string2);

   concat(string1, string2, result);
   printf("Concatenated string: %s\n", result);

   int length = countCharacters(result);
   printf("Length of concatenated string: %d\n", length);

   length = countCharacters(string1);
   printf("Length of string1: %d\n", length);
```

```
    length = countCharacters(string2);
    printf("Length of string2: %d\n", length);

    if (areStringsEqual(string1, string2)) {
        printf("The strings are equal.\n");
    } else {
        printf("The strings are not equal.\n");
    }

    return 0;
}

int countCharacters(const char *str) {
    int count = 0;
    while (*str != '\0') {
        count++;
        str++;
    }
    return count;
}

void concat(char *str1, char *str2, char *res) {

    while (*str1 != '\0') {
        *res = *str1;
        res++;
        str1++;
    }

    while (*str2 != '\0') {
        *res = *str2;
        res++;
        str2++;
    }
    *res = '\0';
}

int areStringsEqual(const char *str1, const char *str2) {
    while (*str1 != '\0' && *str2 != '\0') {
        if (*str1 != *str2) {
            return 0;
        }
        str1++;
        str2++;
    }

    if (*str1 == '\0' && *str2 == '\0') {
        return 1;
    } else {
        return 0;
    }
}
```

Strlen function

```
#include <stdio.h>
#include<string.h>
int main(){
```

```
   char name[]="Abhinav";
   printf("Length of the name is %d",strlen(name));
}
```

Strcpy

```
#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;
}
```