1.WAP to return the array sum by passing array as an argument

```
#include<stdio.h>
int arraySum(int *arr,int n);


int main(){
   int arr_sum=0;
   int array[10]={1,2,3,4,5,6,7,8,9,10};


   arr_sum=arraySum(&array[0],10);
   printf("sum = %d",arr_sum);


}


int arraySum(int *arr,int n){
   int sum=0;
   for(int i=0;i<n;i++){
      sum=sum+*(arr+i);
   }
   return sum;
}
```

OUTPUT

```
sum = 55
```

2. Problem : 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(int const *arr,int n);
int main(){
   int a[10]={1,2,3,4,5,6,7,8,9,10};
   int const *ptr=a;
  printArray(ptr,10);
   return 0;
}
void printArray(int const *arr,int n){
 *(arr+2)=8;
   for(int i=0;i<n;i++){
      printf("a[%d] = %d\n",i, *(arr+i));
   }
  }
```

OUTPUT

```
error: assignment of read-only location '*(arr + 8u)'
    *(arr+2)=8;
          ^
```

3.Problem :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 a=10;
   int b=100;
   int const *ptr = &a;
   int const *const ptr1 = &b;

   printf("value of a = %d\n",*ptr);
   printf("value of pointer = %p\n",ptr);
   // *ptr =19;
   // printf("value of a = %d",*ptr);
   // error: assignment of read-only location '*ptr'
   //  *ptr =19;

   printf("value of b = %d\n",*ptr1);
   printf("value of pointer = %p\n",ptr1);
//    ptr1=&b;
//    printf("value of b = %d\n",*ptr1);
//    printf("value of pointer = %p\n",ptr1);
// error: assignment of read-only variable 'ptr1'
//     ptr1=&b;

}
```

OUTPUT

```
error: assignment of read-only location '*ptr'
```

```
    *ptr =19;
         ^
ass3.c:34:9: error: assignment of read-only variable 'ptr1'
    ptr1=&b;
        ^
```

4.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 integer_data;

  float float_data;

  char sting_data[50];

  printf("Enter the data : (int,float,string)\n");

  scanf("%d %f %[^\n]",&integer_data,&float_data,sting_data);


  print_data(&integer_data,'i');

  print_data(&float_data,'f');

  print_data(&sting_data,'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;

   default:
   printf("invalid type\n");
      break;
   }

}
```

<u>OUTPUT</u>

```
Enter the data : (int,float,string)
40 3.14 hello world
Integer : 40
float : 3.14
string : hello world
```

<u>5.WAP to find length, concatenate and compare 2 strings</u>

```
#include<stdio.h>
void find_count(char str1[],char str2[]);
void concatenate(char str1[],char str2[]);
void equal_or_not(char str1[],char str2[]);
```

```
int main(){
    int op;
    char string1[100];
    char string2[100];
    printf("enter first string\n");
    scanf("%s",string1);

    printf("enter second string\n");
    scanf("%s",string2);

    printf("enter options\n1.count\n2.concatenation\n3.compare\n");
    scanf("%d",&op);

    switch (op)
    {
    case 1:
        find_count(string1,string2);
        break;
    case 2:
        concatenate(string1,string2);
        break;
    case 3:
        equal_or_not(string1,string2);
        break;

    default:
    printf("invalid option");
        break;
    }

}
```

```
void find_count(char str1[],char str2[])
{
    int count = 0;
    int i=0;

    while (str1[i]!='\0')
    {
        count=count+1;
        i++;
    }
    printf("printf count of first string = %d\n",count);
    while (str2[i]!='\0')
    {
       count=count+1;
       i++;
    }

    printf("count of second string = %d\n",count);

}

void concatenate(char str1[],char str2[]){
    int i,j;
    char arr[200];
    for(i=0;str1[i]!='\0';i++){
        arr[i]=str1[i];
    }
    for(j=0;str2[j]!='\0';j++){
        arr[i+j]=str2[j];
    }
    arr[i+j]='\0';
    printf("concatinated string = %s\n",arr);
}

void equal_or_not(char str1[],char str2[]){
    int flag=0,i;
    for (i = 0; str1[i] != '\0' && str2[i] != '\0'; i++) {
        if (str1[i] != str2[i]) {
            flag = 1;
            break;
        }
    }
    if (str1[i] != '\0' || str2[i] != '\0') {
        flag = 1;
    }

    if (flag == 0) {
        printf("Strings are equal.\n");
    } else {
        printf("Strings are not equal.\n");
    }
}
```

OUTPUT

```
enter first string
amritha
enter second string
rajeevan
enter options
1.count
2.concatenation
```

```
3.compare
1
printf count of first string = 7
count of second string = 8

enter first string
amritha
enter second string
 rajeevan
enter options
1.count
2.concatenation
3.compare
2
concatinated string = amritharajeevan

enter first string
malayalam
enter second string
malayalam
enter options
1.count
2.concatenation
3.compare
3
Strings are equal.

enter first string
hello
enter second string
world
```

```
enter options

1.count

2.concatenation

3.compare

3

Strings are not equal.
```