POINTERS AND CONSTANTS

```
#include <stdio.h>

int main()
{
   int num=800;
   printf("001num=%d \n",num);
   const int *pnum=&num;
   num=900;
   printf("001num=%d \n",num);

   return 0;
}
```

```
001num=800
001num=900
```

```
#include <stdio.h>

int main()
{
   int num=800;
   printf("001num=%d \n",num);
   const int *pnum=&num;
   num=900;
   printf("001num=%d \n",num);
   *pnum=500;
   return 0;
```

}

```
main.c: In function ‘main’:
main.c:18:10: error: assignment of read-only location ‘*pnum’
   18 |     *pnum=500;
      |          ^
```

```
/*
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 POINTERS

```
#include <stdio.h>


int main()

{

   int i=1234;

   float pi=3.14;

   char c='A';

   void *ptr;
```

```
    ptr=&i;
    printf("i=%d \n",*ptr);
    //const int *pnum=&num;
    //num=900;
    //printf("001num=%d \n",num);
    //*pnum=500;
    return 0;
}
```

```
main.c: In function ‘main’:
main.c:18:22: warning: dereferencing ‘void *’ pointer
   18 |     printf("i=%d \n",*ptr);
      |                       ^~~~
main.c:18:22: error: invalid use of void expression
```

TYPECASTING VOID POINTERS TO ANY DATATYPE

```
#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=%d \n",*(float *)ptr);
    ptr=&c;
```

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

    return 0;
}
```

```
i=1234
pi=684901024
c=65
```

POINTERS AND ARRAYS

```
#include <stdio.h>

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

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

    return 0;
}
```

```
Address of A[]=0x7ffcadd291ec
```

```
#include <stdio.h>

int main()
{
```

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

    printf("Address of A[0]=%p \n",a);
    printf("Address of A[1]=%p \n",a+1);
    printf("Address of A[2]=%p \n",a+2);

    return 0;
}
```

```
Address of A[0]=0x7ffe7042a0ac
Address of A[1]=0x7ffe7042a0b0
Address of A[2]=0x7ffe7042a0b4
```

Reason we can’t use & in pointers in any array

```
#include <stdio.h>

int main()
{
    int a[]={1,2,3};
    int *ptr=a;
    printf("Address of A[0]=%p \n",a);
    printf("ptr=%p \n",ptr);

    return 0;
}
```

```
Address of A[0]=0x7fff8ceeae0c
ptr=0x7fff8ceeae0c
```

here we can simply write ‘a’ rather than int *ptr=a; because both giving the same in 2 diff ways

```
#include <stdio.h>

int main()
{
   int a[]={1,2,3};
   int *ptr=&a[0];
   printf("Address of A[0]=%p \n",a);
   printf("ptr=%p \n",ptr);

   return 0;
}
```

```
Address of A[0]=0x7ffced13bbdc
ptr=0x7ffced13bbdc
```

```
#include <stdio.h>

int main()
{
   int a[]={1,2,3};
    printf("001The element at the oth index is=%d \n",a[0]);
     printf("002The element at the oth index is=%d \n",*(a+0));
   int *ptr=&a[0];
   printf("Address of A[0]=%p \n",a);
   printf("ptr=%p \n",ptr);
```

```
   return 0;
}
```

```
001The element at the oth index is=1
002The element at the oth index is=1
Address of A[0]=0x7ffdbae7399c
ptr=0x7ffdbae7399c
```

```
#include <stdio.h>

int main()
{
   int a[]={1,2,3};
    printf("Address of A[0]=%p \n",a);
    printf("001The element at the oth index is=%d \n",a[0]);
     printf("002The element at the oth index is=%d \n",*(a+0));
      printf("Address of A[1]=%p \n",a+1);
     printf("001The element at the of 1st index is=%d \n",a[1]);
     printf("002The element at the 1st index is=%d \n",*(a+1));
   //int *ptr=&a[0];
   //printf("Address of A[0]=%p \n",a);
  // printf("ptr=%p \n",ptr);

   return 0;
}
```

```
Address of A[0]=0x7ffe22cc483c
001The element at the oth index is=1
002The element at the oth index is=1
```

Address of A[1]=0x7ffe22cc4840

001The element at the of 1st index is=2

002The element at the 1st index is=2

```
#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];
}
```

```
#include <stdio.h>


int main()

{

    int a[]={1,2,3,4,5,6,7,8,9};

    int *ptr=a;//initialise the pointer with the address of array a[]

   for(int i=0;i<9;i++)

   {

    printf("a[i]=%d -> \n",i,*(ptr+i));

   }

    *(ptr+3)=8;
```

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

```
a[i]=0 ->
a[i]=1 ->
a[i]=2 ->
a[i]=3 ->
a[i]=4 ->
a[i]=5 ->
a[i]=6 ->
a[i]=7 ->
a[i]=8 ->
a[i]=0 ->
a[i]=1 ->
a[i]=2 ->
a[i]=3 ->
a[i]=4 ->
a[i]=5 ->
a[i]=6 ->
```

a[i]=7 ->

a[i]=8 ->

```
#include <stdio.h>

int main()
{
    int a[]={1,2,3,4,5,6,7,8,9};
    printf("Address of a=%p\n",a+1);
    int *ptr=a;//initialise the pointer with the address of array a[] int *ptr=&a[0]
    printf("Address of a[1]=%p\n",ptr+1);
   //reinitialise the pointer to the element present in the ist index
   ptr=&a[1];
    printf("Address of a[1]=%p\n",ptr);
    return 0;
}
```

Address of a=0x7ffe0e2d9044

Address of a[1]=0x7ffe0e2d9044

Address of a[1]=0x7ffe0e2d9044

```
#include <stdio.h>


int main()
{
    int a[]={1,2,3,4,5,6,7,8,9};
    printf("Address of a=%p\n",a+1);
    int *ptr=a;//initialise the pointer with the address of array a[] int *ptr=&a[0]
    printf("Address of a[1]=%p\n",ptr+1);
   //reinitialise the pointer to the element present in the ist index
   ptr=&a[1];
    printf("Address of a[1]=%p\n",ptr);
    printf("Address of a[2]=%p\n",ptr+1);


    return 0;
}
```

```
Address of a=0x7ffdafe6b204
Address of a[1]=0x7ffdafe6b204
Address of a[1]=0x7ffdafe6b204
Address of a[2]=0x7ffdafe6b208
```

Qn.//n represents the number of elements in an 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);
}
int addArray(int array[],int n){
   int arSum=0;
   for(int i=0;i<n;i++){
      arSum=arSum+array[i];
   }
   return arSum;
}
```

output

Sum=45

With pointers

```
#include <stdio.h>

int addArray(int *array, int n);  // Function prototype using pointer
```

```
int main() {
    int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int sum = 0;
    sum = addArray(a, 10);  // Pass the array as a pointer
    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);  // Use pointer arithmetic to access array elements
    }
    return arSum;
}
```

Sum=45

Qn. 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.

ANSWER

```
#include <stdio.h>

int printArray(const int *arr, int size);

int main() {
    int arr[5] = {1, 2, 3, 4, 5};

    const int *ptr = arr;
    printArray(ptr, 5);
```

```
    return 0;
}

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

    printf("The elements in the array before modification:\n");
    for (int i = 0; i < size; i++) {
        printf("%d -> ", *(arr + i));
    }
    printf("\n");

    // Attempt to modify an element (this will cause a compile-time error)
    // Uncommenting the following line will result in an error because arr is a pointer to const int
     //*(arr + 2) = 10;  // This is not allowed
    printf("The elements in the array after attempted modification (should be unchanged):\n");
    for (int i = 0; i < size; i++) {
        printf("%d -> ", *(arr + i));
    }
    printf("\n");

    return 0;
}
```

Output

The elements in the array before modification:

1 -> 2 -> 3 -> 4 -> 5 ->

The elements in the array after attempted modification (should be unchanged):

1 -> 2 -> 3 -> 4 -> 5 ->

```
main.c: In function ‘printArray’:
main.c:30:17: error: assignment of read-only location ‘*(arr + 8)’
   30 |      *(arr + 2) = 10;  // This is not allowed
      |                 ^
```

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;         // Define an integer variable and initialize it with a value
    int anotherValue = 20;  // Another integer variable for demonstration

    // 1. Create a pointer to a const integer
    const int* ptrToConstVal = &value;  // Pointer to a constant integer (changed pointer name)

    // 2. Demonstrate that the value cannot be modified through the pointer
    printf("Using pointer to const integer:\n");
    printf("Value of 'value' through ptrToConstVal: %d\n", *ptrToConstVal);  // Prints the value
    // Attempting to modify the value through the pointer will cause a compile-time error
    // *ptrToConstVal = 15;  // ERROR: cannot modify the value through the pointer to const integer
```

```
    // 3. Create a const pointer to an integer
    int* const constPtr = &value;  // Constant pointer to an integer

    // 4. Demonstrate that the pointer itself cannot be changed
    printf("\nUsing const pointer to integer:\n");
    printf("Value of 'value' through constPtr: %d\n", *constPtr);  // Prints the value

    // Attempting to change the address the const pointer holds will cause a compile-time error
    // constPtr = &anotherValue;  // ERROR: cannot change the address of const pointer
    printf("Address of 'value' (ptrToConstVal points to): %p\n", (void*)ptrToConstVal);
    printf("Address of 'value' (constPtr points to): %p\n", (void*)constPtr);
    value = 30;
    printf("\nAfter modifying 'value' directly:\n");
    printf("Updated value: %d\n", value);
    printf("Address of 'value' (ptrToConstVal points to after modification): %p\n",
(void*)ptrToConstVal);

    return 0;
}
```

```
Using pointer to const integer:
Value of 'value' through ptrToConstVal: 10

Using const pointer to integer:
```

Value of 'value' through constPtr: 10

Address of 'value' (ptrToConstVal points to): 0x7fff95860dc0

Address of 'value' (constPtr points to): 0x7fff95860dc0

After modifying 'value' directly:

Updated value: 30

Address of 'value' (ptrToConstVal points to after modification): 0x7fff95860dc0

STRINGS

```
#include <stdio.h>

int main()
{
    printf("Hi my name \0 is Varsha");

    return 0;
}
```

Hi my name

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

```
int main()
{
    char name[]={'r','o','y'};
    printf("size of name=%d\n",sizeof(name));
    printf("%s",name);

    return 0;
}
size of name=3
roy

#include <stdio.h>

int main()
{
    char name[]={"Varsha"};
    printf("size of name=%d\n",sizeof(name));
    printf("%s",name);

    return 0;
}
size of name=7
```

Varsha

```
#include <stdio.h>

int main()
{
    char name[5]={"Varsha"};
    for(int i=0;i<6;i++)
    {
        printf("%c\n",name[i]);
    }
    printf("size of name=%d\n",sizeof(name));
    printf("%s",name);

    return 0;
}
```

size of name=5

Varsh

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

```
int main()
{
   char name[100];
   name="Varsha";

   return 0;
}
```

```
main.c: In function ‘main’:
main.c:14:9: error: assignment to expression with array type
   14 |     name="Varsha";
      |         ^
```

```
#include <stdio.h>

int main()
{
   char name[]="Varsha";
   printf("%s",name);

   return 0;
}
```

Varsha

```
#include <stdio.h>

int main()
{
   char name[50];
   printf("Enter the name:");
   scanf("%s",name);
   printf("The name is %s",name);
   printf("\n");

   return 0;
}
```

```
Enter the name:Varsha
The name is Varsha
```

```
#include <stdio.h>

int main()
{
        char str1[]="Varsha";
        char str2[]="Venu";
```

```
    int count=0;
    while(str1[count]!='\0')
    {
       count=count+1;
    }

    printf("The length is %d\n",count);
    count=0;
    while(str2[count]!='\0')
    {
       count=count+1;
    }
            printf("The length is %d",count);


    return 0;
}
```

The length is 6

The length is 4

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);
    }
}
```

```
Integer: 42
Float: 3.14
String: Hello, world!
Error: Unsupported type specifier 'x'
```

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

```
#include <stdio.h>

void concatenate(char result[], const char str1[], const char str2[]);

int main() {
    const char str1[] = "Varsha";
    const char str2[] = "Venu";
    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';
}
```

The result of concatenation is: VarshaVenu

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

    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;
}
```

Equal

```
#include <stdio.h>
#include<string.h>
int main()
{
   char name[]="Varsha";
   printf("The length of the name is=%d",strlen(name));

   return 0;
}
```

The length of the name is=6

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

}

The length of the name is=6

Initials=Varsha