POINTERS AND CONST

```
/*
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",*(int *)ptr);


   ptr=&pi;

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


   ptr=&c;

   printf("c=%c",*(char *)ptr}
```

Output:

i=1234

f=3.140000

c=A

POINTERS AND ARRAYS

```
#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); //Address jump
   printf("Address of a[2] = %p\n",a+2);



}
#include<stdio.h>
int main(){
   int a[]={1,2,3};
   int *ptr=a; //need not use '&'
   printf("Address of a[0] = %p\n",a);
   printf("ptr = %p\n",ptr);



}
#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);

   }
```

```
#include<stdio.h>
int main(){
   int a[]={1,2,3};
   printf("Address of a[0] = %p\n",a);
   printf("the element at the 0th index = %d\n",a[0]);
   printf("the element at the 0th index = %d\n",*(a+0));
   printf("Address of a[1] = %p\n",a+1);
   printf("the element at the 1st index = %d\n",a[1]);
   printf("the element at the 1st index = %d\n",*(a+1));
   int *ptr=&a[0];

}
```

Output:

```
Address of a[0] = 0x7ffcca809c7c
the element at the 0th index = 1
the element at the 0th index = 1
Address of a[1] = 0x7ffcca809c80
the element at the 1st index = 2
the element at the 1st index = 2
```

```
#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))}}
```

POINTER ARITHMETIC

```
#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[] int *ptr=&a[0]
   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);
}
```

Output:

```
Address of a[1] = 0x7ffcf5727814
Address of a[1] = 0x7ffcf5727814
Address of a[1] = 0x7ffcf5727814
Address of a[2] = 0x7ffcf5727818
```

Passing array to function

```
// n represents 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("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];
    }
    return arSum;
}
```

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[0],10);
  printf("Sum = %d \n",sum);
  return 0;
}
int addArray(int *array,int n){
    //int *ptr=array;
    int arSum=0;
    for(int i=0;i<n;i++){
        arSum+= *(array+i);
    }
    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>
int printArray(const int *arr,int size);
int main(){
   int const array[5]={1,2,3,4,5};
   int const *parray=array;
   //*(parray + 3)=8;
   printArray(array,5);
}
int printArray(const int *arr,int size){
   for(int i=0;i<size;i++){
      printf("array[%d] = %d",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;
   int const *ptr1=&value;
   printf("value = %d\n",*ptr1);
   printf("Address of ptr is %p\n",ptr1);
   //*ptr1=20;
   //printf("value = %d",*ptr1);
   int num1=15;
   ptr1=&num1;
   printf("Address of ptr1 is %p\n",ptr1);

   int num2=30;
   printf("num2 = %d\n",num2);
   int *const ptr2=&num2;
   *ptr2=35;
```

```
    printf("num2 = %d\n",*ptr2);
    int num4=60;
    //ptr2=&num4;
}
```

STRING

Outputting:

```
#include<stdio.h>
int main(){
    char name[] = "Anjali";
    printf("Name = %s\n",name);
    return 0;
}
```

Inputting:

```
#include<stdio.h>
int main(){
    char name[10];
    printf("Enter your name: ");
    scanf("%s",name);
    printf("\n");
    printf("The name is %s",name);
    return 0;
}
```

Length of string

```
#include<stdio.h>
int main(){
    char str1[] = "My name is Anjali";
    char str2[] = "Hello world";
    int count=0;
    while(str1[count]!='\0')
        ++count;
    printf("Length of str1 is %d\n",count);
```

```
    count=0;
    while(str2[count]!='\0')
        count++;
    printf("Length of str2 is %d",count);
    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 integer;
   float fl;
   char string[50];
   char type1='i';
   char type2='f';
   char type3='s';
   scanf("%d %f %s",&integer,&fl,string);
   print_data(&integer,'i');
   print_data(&fl,'f');
   print_data(&string,'s');
}


void print_data(void *data,char type){
   switch (type) {
      case 'i': {
         int* int_data = (int*)data;
         printf("Integer: %d\n", *int_data);
         break;
      }
```

```
        case 'f': {
            float* float_data = (float*)data;
            printf("Float: %.2f\n", *float_data);
            break;
        }
        case 's': {
            char* string_data = (char*)data;
            printf("String: %s\n", string_data);
            break;
        }
        default:
            printf("Error: Unsupported type specifier\n");
            break;
}
}
```

String assignment (Length, concatenation, comparison)

```
#include<stdio.h>
#include<stdbool.h>
int count(char str1[]);
void concat(const char str1[],const char str2[],char result[]);
bool equal(const char str1[],const char str2[]);
int main(){
    char str1[50];
    char str2[50];
    char r[50];
    printf("Enter the first string: ");
    scanf("%s",str1);
    printf("Enter the second string: ");
    scanf("%s",str2);
    int c = count(str1);
```

```
    printf("Length of string1 is %d\n",c);

    concat(str1,str2,r);
    printf("Concatenated string is %s\n",r);



    if(equal(str1,str2)){
        printf("Strings are equal");
    }
    else{
        printf("Strings are not equal");
    }
}
int count(char str1[]){
    int count=0;
    while(str1[count]!='\0')
        ++count;
    return count;

}
void concat(const char str1[],const char str2[],char result[]){
    int i=0;
    while(str1[i]!='\0'){
        result[i]=str1[i];
        i++;
    }
    int j=0;
    while(str2[j]!='\0'){
        result[i]=str2[j];
        j++;
        i++;
```

```
    }
}
bool equal(const char str1[],const char str2[]){
    int i=0;
    while (str1[i] != '\0' && str2[i] != '\0') {
        if (str1[i] != str2[i]) {
            return false;
        }
        i++;
    }
    if (str1[i] == '\0' && str2[i] == '\0') {
        return true;
    }

}
```

String functions

Strlen:

```
#include <stdio.h>
#include <string.h>
int main(){
    char name[] ="Abhinav";

    printf("The length of the name is = %d",strlen(name));

    return 0;
}
```

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;
}
```