DAY 9

1)

```
#include <stdio.h>

int main()
{
    int num=800;
    printf("001num=%d\n",num);
    const int *pnum=&num;
    *pnum=900;
    printf("002num=%d\n",num);

    return 0;
}
```

2)//int const*===>value becomes constant but the pointer is modifible
//int *const==>value becomes modifiable but the pointer becomes constant

```
#include <stdio.h>

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

```
    int num1=600;
    *pnum=&num1;
    //*pnum=900;
    //printf("002num=%p\n",pnum);
    printf("002num=%d\n",num);
    printf("002num=%d\n",pnum);
    return 0;
}


3)/*
 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;
 }
```

4)void pointer

```
#include <stdio.h>

int main()
{
    int i=1234;
    float pi=3.14;
    char c='A';
    void *ptr;
    ptr=&i;
    printf("i=%d",*(int*)ptr);

    return 0;
}
```

5)

```
#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;
}
```

POINTERS AND ARRAY

1)

```
#include <stdio.h>

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

    printf("address of A=%p\n",a);

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

    printf("address of A=%p\n",a+2);
```

```
    return 0;
}
```

2)

```
#include <stdio.h>

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

    //int *ptr=a;

    int *ptr=&a[0];

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

    return 0;
}
```

6)

```
#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];

}
```

POINTER ARTHIMETIC

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

int main(){

    int a[] ={1,2,3,4,5,6,7,8,9};
    int *ptr=a;
    for(int i=0;i<9;i++){
        printf("a[%d]=%d->",i,*(ptr+i));
    }
    printf("\nUPDATED ARRAY\n");
    *(ptr+3)=8;
     for(int i=0;i<9;i++){
        printf("a[%d]=%d->",i,*(ptr+i));
    }

}
```

a[0]=1->a[1]=2->a[2]=3->a[3]=4->a[4]=5->a[5]=6->a[6]=7->
a[7]=8->a[8]=9->
UPDATED ARRAY
a[0]=1->a[1]=2->a[2]=3->a[3]=8->a[4]=5->a[5]=6->a[6]=7->
a[7]=8->a[8]=9->

8)

```
#include <stdio.h>

int arraySum(int *arr, int n);  // Function prototype to
calculate the sum

int main() {
    int n;

    // Input size of the array
    printf("Enter size of the array: ");
    scanf("%d", &n);

    // Declare the array with the specified size
    int arr[n];  // Variable Length Array (VLA)

    // Input array elements
    printf("Enter array elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Print array elements
```

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

    // Calculate sum using the arraySum function
    int sum = arraySum(arr, n);
    printf("Sum = %d\n", sum);

    return 0;
}

// Function to calculate the sum of array elements
int arraySum(int *arr, int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += *(arr + i);  // Access array elements using
pointer arithmetic
    }
    return sum;  // Return the computed sum
}
```

Or

```
#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); //&a[0]
    printf("the sum is %d",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;
}
```

ASSIGNMENT

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

```
    int const *ptr=&arr[0];
    *ptr=90;
    int x=printArray(arr,5);



    return 0;
}
int printArray(const int *arr,int n){
    for(int i=0;i<n;i++){
        printf("%d\n",*(arr+i));
}
}
```

2)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 x=10;
   //int const *ptr;
```

```
   int *const ptr=&x;
  // *ptr=70;
  printf("address of ptr=%p\n",ptr);
  printf("address of num=%p\n",&x);
   printf("value of x=%d\n",*ptr);
   int num1;
   //ptr=&num1;
   //printf("address of ptr=%p",ptr);

  return 0;
}
```

STRING

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

  char name[]={"navya"};
  for(int i=0;i<6;i++){
      printf("%c",name[i]);
  }


  printf("size=%d\n",sizeof(name));
  //printf("my name is navya");

  return 0;
}
2)#include<stdio.h>
```

```
int main(){

  char str1[]="how are you";
  char str2[]="hello";
  int count=0;
  int count1=0;
  int i=0;
  while(str1[i]!='\0')
  {
      count=count+1;
      i++;
  }
  printf("length =%d\n",count);


     while(str2[i]!='\0')
  {
      count1=count1+1;
      i++;
  }
  printf("length =%d",count1);



  return 0;
}
```

ASSIGNMENT

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 i=42;
    float f=3.14;
    char s[]="hello world!";
    printf("input data:%d(int),%.2f(float),%s(string)\n",i,f,s);
    print_data(&i,'i');
    print_data(&f,'f');
    print_data(&s,'s');
    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("unsupported type specifier is passed");
    }
}
```

3)STRING CONCATENATE

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

```
void concatenate(char str1[],char str2[],char result[]);
int main()
{
   char str1[10]="hello";
   char str2[10]="world";
   char result[20];
   concatenate(str1,str2,result);
  printf("result=%s",result);

    return 0;
}
void concatenate(char str1[],char str2[],char result[]){

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

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

4)strng comparison

```
#include<stdio.h>
int my_strcmp( char str1[], char str2[]);
int main()
```

```
{

    char str1[]="World";
    char str2[]="World";

    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;
}
```

STRING FUNCTIONS

1)

```
#include<stdio.h>
#include<string.h>
int main(){
    char name[]="navya";
    printf("the lengh of name=%d",strlen(name));
    return 0;
}
```

2)