**PHASE 1 DAY 9**

**Pointers**

```
/*
 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 pointer**

```
//Void pointers(3 cases)

#include <stdio.h>

int main()
{
   int i=1234;
   float pi=3.14;
   char c='A';

   void * ptr;//point to any data type
```

```
   /*
    ptr=&i;
    printf("i = %d",*ptr);//warning: dereferencing ‘void *’ pointer
    printf("i = %d",*(int *)ptr);
  */

   /*
    ptr=&c;
    //printf("c = %c",*ptr);//warning: dereferencing ‘void *’ pointer
    printf("c = %c",*(char *)ptr);
    */


    ptr=&pi;
    //printf("pi = %f",*ptr);//warning: dereferencing ‘void *’ pointer
    printf("pi = %f",*(float *)ptr);


        return 0;

}
```

**POINTER AND ARRAYS**

1.//Pointer to array

```
#include <stdio.h>

int main()
{
    int a[]={1,2,3};
    int*ptr=a;
    int *ctr=&a[0];

   printf("Adress a = %p\n",ptr);
   printf("Adress a[0] = %p",ctr);
        return 0;
}
```

2.#include <stdio.h>

```
int main()
{
    int a[]={1,2,3};
    printf("001 element of 0th index %d\n",a[0]);
    printf("002 element of 0th index %d\n",*(a+0));
    printf("003 element of 0th index %d\n",a[1]);
    printf("004 element of 0th index %d\n",*(a+1));  //Jumps to next element

    int*ptr=a;
    return 0;
}
```

3.//Array to function

```
#include <stdio.h>
int sum (int *array,int n);

int main()
{
    int A[11]={0,1,2,3,4,5,6,7,8,9};
    int Sum=0;

   Sum = sum(A,10);
   printf("Sum=%d",Sum);
    return 0;
}


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

```
}
//Array to function

#include <stdio.h>
int sum (int * array,int n);

int main()
{
    int A[11]={1,2,3,4,5,6,7,8,9,10};
    int Sum=0;

   Sum = sum(A,10);
   printf("Sum=%d",Sum);
    return 0;
}

int sum(int arr[],int n);
{
    int arraysum=0;
    for(int i=0;i<n;i++)
    {
        arraysum+=array[i];
    }
    return arraysum;
}
```

**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:**

```
//Array to function
#include <stdio.h>
void printArray(const int *arr,int size);
int main()
{
    int A[5]={1,2,3,4,5};
   printArray(ptr,5);
    return 0;
}
void printArray(const int*arr,int size)
{
```

```
    for(int i=0;i<size;i++)
    {
        printf("a[%d]->%d\n",i,*(arr+i));
    }
    //*(arr+4)=10;    //error: assignment of read-only location '*(arr + 16)'
}
```

**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.**

```
//Array to function

#include <stdio.h>

int main()
{
    //Case 1
    /*
    const int value=10;
    int const *ptr=&value;
    printf("001 value %d",value);
    printf("pointer address:%p",ptr);
    value =20; //assignment of read-only location ‘*ptr’
  */
  //Case 2
   /*
    int value=10;
   int* const ptr= &value;
    printf("002 value %d",value);
    printf("pointer address:%p",ptr);
    ptr = 90;
    */
    return 0;
}
```

**STRINGS**

**1.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').**

**Answer:**

```
#include <stdio.h>
void print_data(void* data, char type);

int main()
{
    int int_var = 42;
    float float_var = 3.14;
    char* string_var = "Hello, world!";

    print_data(&int_var, 'i');
    print_data(&float_var, 'f');
    print_data(string_var, 's');
    print_data(&int_var, 'x');

    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("Error: Unsupported type specifier '%c'\n", type);
            break;
```

```
   }
}
```

2.

**Concatenation,stringlength,string equal functions without using string functions**

```
#include <stdio.h>
int string_equal(const char*, const char*);
void concatenate(const char*, const char*);
int string_length(const char * string);

int main()
{
   char const *str1="Nanda ";
   char const*str2="Prasanth";
   concatenate(str1,str2);

   int result=string_equal(str1,str2);

   if(0==result)
   {
      printf("Strings are equal\n");
   }
   else
   {
      printf("String not equal\n");
   }
    return 0;
}

void concatenate(const char* string1, const char*string2)
 {
     //length of both strings
    int len1=string_length(string1);
    int len2=string_length(string2);
     printf("length of string1=%d\nlength of second string= %d\n",len1,len2);

   char result[len1+len2+1];
```

```
   for(int i=0;i<len1;i++)
   {
      result[i]=string1[i];
   }


   for(int i=0;i<len2;i++)
   {
      result[len1+i]=string2[i];
   }

   result[len1+len2]='\0';

   printf("Concatenated string is %s\n",result);

}


int string_equal(const char * str1, const char *str2)
 {
       while (*str1 != '\0')
       {
        if (*str1 != *str2)
        {
            return (*str1 - *str2);

        }
        str1++;
        str2++;
    }

    if (*str2 != '\0')
    {
        return -(*str2);
    }

    return 0;
}
```

```
 int string_length(const char * string)
{
   int count=0;
   for(int i=0;string[i]!='\0';i++)
   {
      count += 1;
   }
   return count;
}
```

4.

```
#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;
}
```