Day 9

1. #include<stdio.h>

```
int main()
{
   int num=900;
   // int const *pnum=&num //here modifiable pointer
    int const*const pnum=&num;//constant pointer
    int num1=800;
    pnum=&num1;
    *pnum=800;
    printf("%d",num);
}
```

2. //void pointer

```
#include<stdio.h>
int main()
{
   int i=10;
   float f=2.34;
   char ch='k';
   void *vptr;
   vptr=&i;
   printf("value of i %d \n",*(int *)vptr);
   vptr=&f;
   printf("value of i %f \n",*(float *)vptr);

   vptr=&ch;
   printf("value of i %c \n",*(char *)vptr);

}
```

```
PS D:\learning c\output> & .\'day9-2.exe'
value of i 10
value of i 2.340000
value of i k
PS D:\learning c\output>
```

3. #include<stdio.h>

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

}
```

```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day9-3.exe'
adress of A[0]=0061FF14
adress of A[1]=0061FF18
adress of A[2]=0061FF1C
PS D:\learning c\output>
```

4.

```
#include<stdio.h>
int main()
{
   int values[5];
   int *ptr;
   //int *ptr=values   this is also fine
   //ptr=values;
   //or
   ptr=&values[0];
   printf(" value of ptr is  %p \n",ptr);//ie adress of values[0]
   printf("adress of values[0] =%p",values);
```

}

```
PS D:\learning c\output> & .\'day9-4.exe'
 value of ptr is  0061FF08
adress of values[0] =0061FF08
PS D:\learning c\output>
```

5.
```c
#include<stdio.h>

int main()
{
    int values[5]={1,2,3};
    int *ptr=values[0];
    printf("001 element at index o %d \n",values[0]);
    printf("001 element at index o %d \n",*(values+0));

    printf(" value of ptr is  %p \n",ptr);//ie adress of values[0]
    printf("adress of values[0] =%p",values);
}
```

```
PS D:\learning c\output> & .\'day9-5.exe
001 element at index o 1
001 element at index o 1
 value of ptr is  00000001
adress of values[0] =0061FF08
PS D:\learning c\output>
```

6.
```c
#include<stdio.h>

int main()

{

   int values[]={1,2,3,4,5};

   int *ptr=values;

   printf("address of a[1] %p \n",values+1);

   printf("address of a[1] %p \n",ptr+1);


   //ptr=&values[1];

   //printf("address of a[1] %p \n",ptr);


   for(int i=0;i<4;i++)

   {

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

   }
```

```
    printf("\n");
    *(ptr+3)=8;
    for(int i=0;i<4;i++)
    {
        printf("a[%d] =%d \n",i,*(ptr+i));
    }

}
```

```
address of a[1] 0061FF04
address of a[1] 0061FF04
a[0] =1
a[1] =2
a[2] =3
a[3] =4

a[0] =1
a[1] =2
a[2] =3
a[3] =8
PS D:\learning c\output>
```

7.
```
#include<stdio.h>
int main()
{
    int values[]={1,2,3,4,5};
    int *ptr=values;
    printf("address of a[1] %p \n",values+1);
    printf("address of a[1] %p \n",ptr+1);

    ptr=&values[1];
    printf("address of a[1] %p \n",ptr);
}
```

```
sum =15
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day9-7.exe'
address of a[1] 0061FF0C
address of a[1] 0061FF0C
address of a[1] 0061FF0C
PS D:\learning c\output>
```

8. #include<stdio.h>

```
int addArray(int values[],int n);
//n is number of elemnets of array
//when array is passed as para in fn , sec para should be no.of elements in array
int main()
{
   int values[5]={1,2,3,4,5};
   int sum=0;
   sum=addArray(values,5);
   printf("sum =%d \n",sum);
}
int addArray(int values[],int n)
{
   int arsum=0;
   for(int i=0;i<n;i++)
   {
     arsum=arsum+values[i];
   }
   return arsum;
}
```

```
intials =Abhinav
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day9-8.exe'
sum =15
PS D:\learning c\output>
```

9. #include<stdio.h>

```
int addArray(int *pvalue,int n);
```

```
//n is number of elemnets of array
//when array is passed as para in fn , sec para should be no.of elements in array
int main()
{
   int values[5]={1,2,3,4,5};
  int *pvalue=values;
   int sum=0;
   sum=addArray(pvalue,5);
   printf("sum =%d \n",sum);
}
int addArray(int *pvalue,int n)
{
   int arsum=0;
   for(int i=0;i<n;i++)
   {
     arsum=arsum+*(pvalue+i);
   }
   return arsum;
}
```

```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day9-9.exe'
sum =15
PS D:\learning c\output>
```

10.
```
#include<stdio.h>
int addArray(int *values,int n);
//n is number of elemnets of array
//when array is passed as para in fn , sec para should be no.of elements in array
int main()
{
```

```
    int values[5]={1,2,3,4,5};


    int sum=0;
    sum=addArray(values,5);
    printf("sum =%d \n",sum);
}
int addArray(int *values,int n)
{
    int arsum=0;
    for(int i=0;i<n;i++)
    {
      arsum=arsum+*(values+i);
    }
    return arsum;
}
```

```
PS D:\learning c\output> & .\'day9-13.exe'
PS D:\learning c\output> cd 'd:\learning c\output
PS D:\learning c\output> & .\'day9-10.exe'
sum =15
PS D:\learning c\output>
```

11.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.

```
//array element access
#include<stdio.h>
void printArray(const int*arr ,int size);
int main()
{
    int  arr[5]={1,2,3,4,5};
    printArray(arr,5);



}
void printArray(const int*arr ,int size)
{     arr[0]=1;     expression must be a modifiable lvalue
     for( int i=0;i<size;i++)
     {
       printf("%d",arr[i]);
     }

}
```

12.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.

```
//protecting a value;
#include<stdio.h>
int main()
{
    int value=10;
    const int *const pvalue=&value;
    *pvalue=30;    expression must be a modifiable lvalue
     int value1=20;
     pvalue=&value1;    expression must be a modifiable lvalue

}
```

13. #include<stdio.h>

int main()

{

char a="k";

return 0;

printf("%c",a);

}

14. #include<stdio.h>

int main()

```
{
   printf("hi iam rinta \0 maria");
   char name[]={"rinta"};
   char s[100]="rinta";
   printf("%s",name);
  // error  s="rinra" ;

   for(int i=0;i<6;i++)
   {
      printf("%c \n ",name[i]);
   }
   printf("size of name is %d",sizeof(name));
}
```

```
hi iam rinta rintar
 i
 n
 t
 a
```

15. #include<stdio.h>

```
int main()
{
  char str1[]="To be or not to be";
  char str2[]="that is the question";
  unsigned int count=0;


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

```
  printf("length of string 1 is %d \n",count);
  count=0;

 while(str2[count]!='\0')
 {
  count++;
 }
 count++;
 printf("length of string 2 is %d ",count);



}
```

```
PS D:\learning c\output> & .\'day9-15.exe
length of string 1 is 19
length of string 2 is 21
PS D:\learning c\output>
```

16.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').

15:26has context menu

```
//universal data printer

#include<stdio.h>

void print_data(void* ,char );

int main()

{

   char type;

   void *data;

   int int_data=42;

   float f_data=3.14;

   char str_data[]="hello";

   print_data(&int_data, 'i');

   print_data(&f_data, 'f');

   print_data(str_data, 's');
```

```
    print_data(&int_data, '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;
    }
}
```

```
PS D:\learning c\output> & .\'day9-16.exe'
Integer: 42
Float: 3.14
String: hello
Error: Unsupported type specifier 'x'
PS D:\learning c\output>
```

17.

Requirements

- In this challenge, you are going to write a program that tests your understanding of char arrays

- write a function to count the number of characters in a string (length)
  - cannot use the strlen library function
  - function should take a character array as a parameter
  - should return an int (the length)

- write a function to concatenate two character strings
  - cannot use the strcat library function
  - function should take 3 parameters
    - char result[]
    - const char str1[]
    - const char str2[]
    - can return void

- 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

```
#include <stdio.h>

int string_length(const char []);

void string_concatenate(char [], const char [], const char []) ;

int strings_are_equal(const char [], const char []) ;



int main() {

   char str1[] = "Hello";

   char str2[] = "World";

   char result[100];



   printf("Length of '%s': %d\n", str1, string_length(str1));

   printf("Length of '%s': %d\n", str2, string_length(str2));



   string_concatenate(result, str1, str2);

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

```
    if (strings_are_equal(str1, str2)) {
        printf("Strings '%s' and '%s' are equal.\n", str1, str2);
    } else {
        printf("Strings '%s' and '%s' are not equal.\n", str1, str2);
    }

    if (strings_are_equal("Hello", "Hello")) {
        printf("Strings 'Hello' and 'Hello' are equal.\n");
    } else {
        printf("Strings 'Hello' and 'Hello' are not equal.\n");
    }

    return 0;
}
int string_length(const char str[]) {
    int length = 0;
    while (str[length] != '\0') {
        length++;
    }
    return length;
}

void string_concatenate(char result[], const char str1[], const char str2[]) {
    int i = 0, j = 0;


    while (str1[i] != '\0') {
        result[i] = str1[i];
        i++;
```

```
    }



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



    result[i] = '\0';
}

int strings_are_equal(const char str1[], const char str2[]) {
    int i = 0;

    while (str1[i] != '\0' && str2[i] != '\0') {
        if (str1[i] != str2[i]) {
            return 0; // Strings are not equal
        }
        i++;
    }



    if (str1[i] == '\0' && str2[i] == '\0') {
        return 1;
    }
    return 0;
}
```

```
Length of 'Hello': 5
Length of 'World': 5
Concatenated string: HelloWorld
Strings 'Hello' and 'World' are not equal.
Strings 'Hello' and 'Hello' are equal.
PS D:\learning c\output>
```

18.

```
#include <stdio.h>

#include <string.h>

int main(){

   char name[] ="Abhinav";

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

   return 0;

}
```

```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day9-18.exe'
The length of the name is = 7
PS D:\learning c\output>
```

19.

```
#include <stdio.h>

#include <string.h>

int main(){

   char name[] ="Abhinav";

   char intials[20];

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

   strcpy(intials,name);

   printf("intials =%s",intials);

   return 0;

}
```

```
PS D:\learning c\output> & .\'day9-19.exe'
The length of the name is = 7
intials =Abhinav
PS D:\learning c\output>
```