POINTERS:

A pointer can be used to store the **memory address of** another variable of same variables. A pointer is defined as a derived data type. We can access and manipulate the data stored in that memory location using pointers.

Benefit of using pointers

- ❖ Pointers are more efficient in handling Array and Structure.
- ❖ Pointer allows references to function and thereby helps in passing of function as arguments to other function.
- ❖ It reduces length and the program execution time.
- ❖ It allows C to support dynamic memory management.

Declaration of Pointer

```
data_type* pointer_variable_name;

int* p;
```

Note: void type pointer works with all data types, but isn't used often.

Initialization of Pointer variable

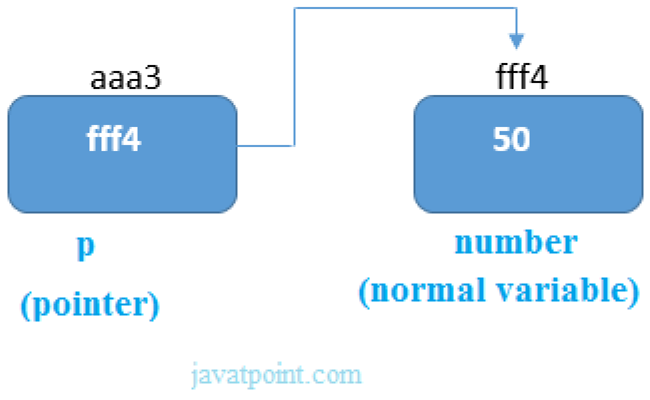
Pointer Initialization is the process of assigning address of a variable to **pointer** variable. Pointer variable contains address of variable of same data type

```
int a = 50 ;
int *p ; //pointer declaration
p = &a ; //pointer initialization
```

Prasant Dash
7008191907

Note:Pointer variable always points to same type of data.

```
float a;
int *ptr;
ptr = &a; //ERROR, type mismatch
```



As you can see in the above figure, pointer variable stores the address of number variable i.e. fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of * (**indirection operator**), **we can print the value of pointer variable p.**

Reference operator (&) and Dereference operator (*)

& is called reference operator. It gives you the address of a variable. There is another operator that gets you the value from the address, it is called a dereference operator (*).

Symbol	**Name**	**Description**
& (ampersand sign)	address of operator	Determines the address of a variable.
* (asterisk sign)	indirection operator	Accesses the value at the address.

Dereferencing of Pointer

Once a pointer has been assigned the address of a variable. To access the value of variable, pointer is dereferenced, using the **indirection operator** *.

```
int a,*p;
a = 10;
p = &a;
printf("%d",*p); //this will print the value of a.
printf("%d",*&a); //this will also print the value of a.
printf("%u",&a); //this will print the address of a.
printf("%u",p); //this will also print the address of a.
printf("%u",&p); //this will also print the address of p.
```

KEY POINTS TO REMEMBER ABOUT POINTERS IN C:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer always be a whole number i.e. address.
- Always C pointer is initialized to null, i.e. int *p = null.
- The value of null pointer is 0.
- & symbol is used to get the address of the variable.
- * Symbol is used to get the value of the variable that the pointer is pointing to.
- If a pointer in C is assigned to NULL, it means it is pointing to nothing.
- Two pointers can be subtracted to know how many elements are available between these two pointers.
- But, Pointer addition, multiplication, division are not allowed.
- The size of any pointer is 2 byte (for 16 bit compiler).

Experiments #01

```
#include <stdio.h>
void main()
{
int number=50;
int *p;
p=&number;//stores the address of number variable
printf("Address of number variable is %x \n",&number);
printf("Address of p variable is %x \n",p);
printf("Value of p variable is %d \n",*p);
        }
```

Experiments #02

```
#include <stdio.h>
int main()
{
int var =10;
int *p;
p= &var;
printf ( "\n Address of var is: %u", &var);
printf ( "\n Address of var is: %u", p);
printf ( "\n Address of pointer p is: %u", &p);
/* Note I have used %u for p's value as it should be an address*/
printf( "\n Value of pointer p is: %u", p);
printf ( "\n Value of var is: %d", var);
printf ( "\n Value of var is: %d", *p);
printf ( "\n Value of var is: %d", *( &var));
}
```

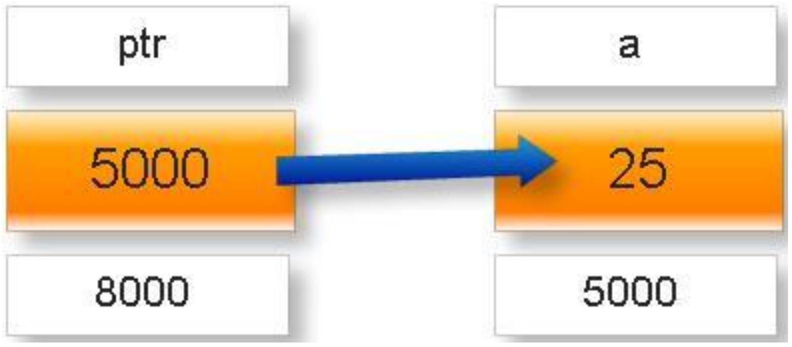
Q1**What is dangling pointer in c?**

Explanation:

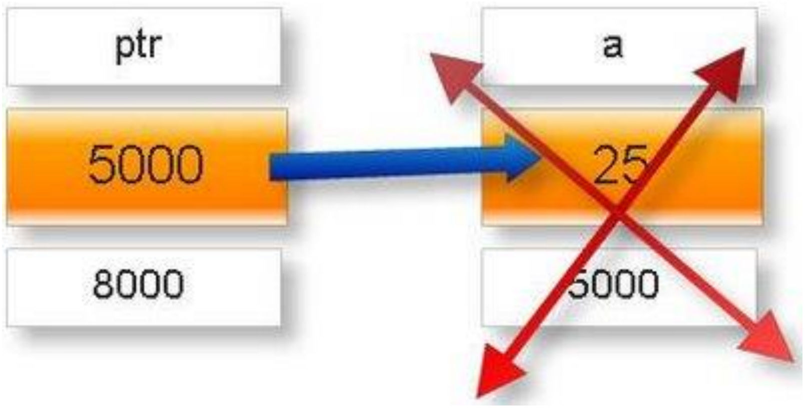
Dangling pointer:

If any pointer is pointing the memory address of any variable but after some variable has deleted from that memory location while pointer is still pointing such memory location. Such pointer is known as dangling pointer and this problem is known as dangling pointer problem.

Initially:



Later:



Prasant Dash
7008191907

Q2**What is wild pointer in c?**

A pointer in c which has not been initialized is known as wild pointer.

Example:

What will be output of following c program?

```
int main(){
        int *ptr;
        printf("%u\n",ptr);
        printf("%d",*ptr);
        return 0;
}
```

Output: Any address

Garbage value

Here ptr is wild pointer because it has not been initialized. There is difference between the NULL pointer and wild pointer. Null pointer points the base address of segment while wild pointer doesn't point any specific memory location.

Q3What is NULL pointer?

Literal meaning of NULL pointer is a pointer which is pointing to nothing. NULL pointer points the base address of segment.

Examples of NULL pointer:

```
int *ptr=(char *)0;
float *ptr=(float *)0;
```

```
char *ptr=(char *)0;
double *ptr=(double *)0;
char *ptr='\0';
int *ptr=NULL;
```

What is meaning of NULL?

Answer:

NULL is macro constant which has been defined in the heard file stdio.h, alloc.h, mem.h, stddef.h and stdlib.h as

```
#define NULL 0
```

Q4.What is size of void pointer?

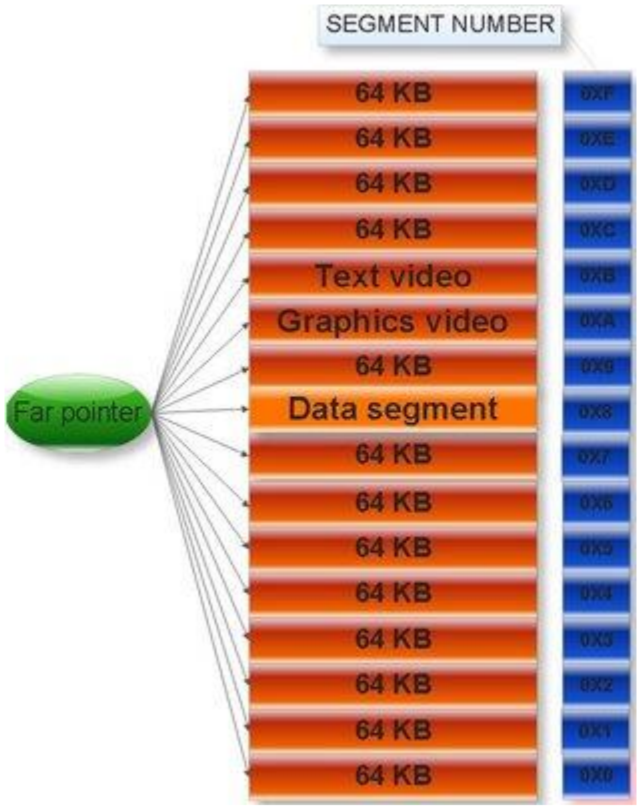
Size of any type of pointer in c is independent of data type which is pointer is pointing i.e. size of all type of pointer (near) in c is two byte either it is char pointer, double pointer, function pointer or null pointer. Void pointer is not exception of this rule and size of void pointer is also two byte.

Q5.What is difference between uninitialized pointer and null pointer?

An uninitialized pointer is a pointer which points unknown memory location while null pointer is pointer which points a null value or base address of segment. For example:

Q6What is the far pointer in c?

The pointer which can point or access whole the residence memory of RAM i.e. which can access all 16 segments is known as far pointer.



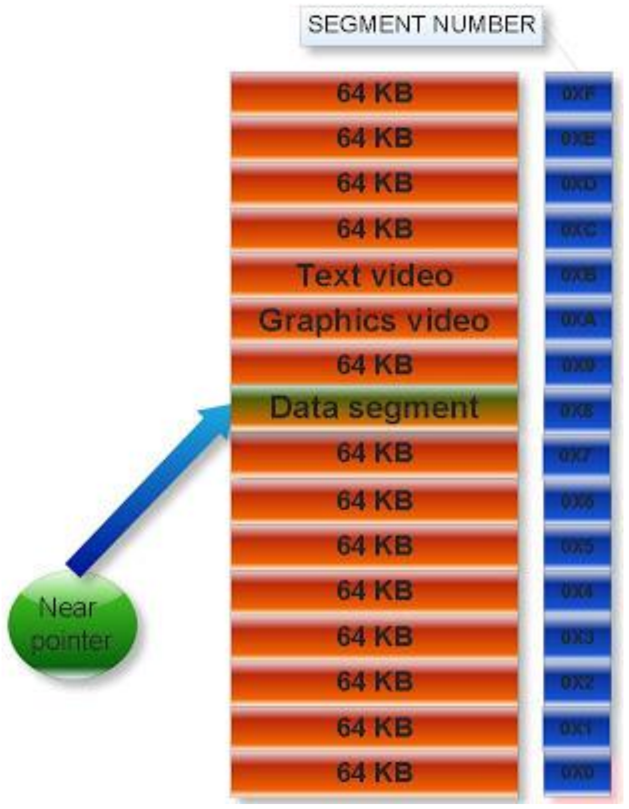
Size of far pointer is 4 byte or 32 bit. Examples:

(1) What will be output of following c program?

```
int main(){
        int x=10;
        int far *ptr;
        ptr=&x;
        printf("%d",sizeof ptr);
        return 0;
}
```

Near pointer:

The pointer which can points only 64KB data segment or segment number 8 is known as near pointer.



That is near pointer cannot access beyond the data segment like graphics video memory, text video memory etc. Size of near pointer is two byte. With help keyword near, we can make any pointer as near pointer.

Examples:

(1)

```
#include<stdio.h>

int main(){

        int x=25;
        int near* ptr;

        ptr=&x;
        printf("%d",sizeof ptr);

return 0;
}
```

Output: 2

Near Pointer :- The pointer which can handle only 1 segment of of 1 mb data is called near pointer.Near Pointer doesn't points to any other segments except data segments

Near Pointer doesn't points to any other segments except data segments.

The size of near pointer is 2 bytes.

By default any type of pointer is near only.

When we are increments the near pointer value then it increase offset address only.

When we are applying the relational operators on near pointer then it compare offset address only. %p %x %u only specifier we can use.

Far Pointer:- Pointer variable which can handle any segment of 1 MB data is called far pointer.

When we are working with far pointer , it can handle any segment from the range ox0 to oxF, but at a time only one segment.

Size is 4 bytes, because it holds segment & offset address also.

When we are increments the near pointer value then it increase offset address only.

When we are applying the relational operators on far pointer then it compare segment address along with offset address .

We required to use %lp, %lu format specifiers for printing the far pointer address.

By using far keyword we can create far pointer.

Huge pointer :- Pointer variable which can handle any segment of 1 MB data is called huge pointer.

When we are working with far pointer , it can handle any segment from the range ox0 to oxF, but at a time only one segment.

Size is 4 bytes, because it holds segment & offset address also.

When we are increments the near pointer value then it increase segment address along with offset address.

When we are comparing the huge pointers, it compares normalization value.

We required to use %lp, %lu format specifiers for printing the huge pointer address.

Normalization :- It is a process of converting 32 bit physical bit into 20 bit hexadecimal format.

Pointers to Pointers

Pointers can point to other pointers /pointer refers to the address of another pointer.

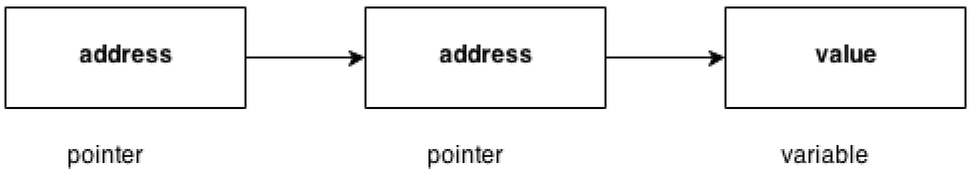
Pointer can point to the address of another pointer which points to the address of a value.

syntax of pointer to pointer

```
int **p2;
```

pointer to pointer example

Let's see an example where one pointer points to the address of another pointer.



Prasant Dash
7008191907

```
#include <stdio.h>
void main()
{
int number=50;
int *p;//pointer to int
int **p2;//pointer to pointer
p=&number;//stores the address of number variable
p2=&p;
printf("Address of number variable is %x \n",&number);
printf("Value of *p variable is %d \n",*p);
printf("Address of p2 variable is %x \n",p2);
printf("Value of **p2 variable is %d \n",**p);
        }
```

Arrays and Pointers

When an array is declared, compiler allocates sufficient amount of memory to contain all the elements of the array. Base address which gives location of the first element is also allocated by the compiler.

Suppose we declare an array arr,

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

Assuming that the base address of **arr** is 1000 and each integer requires two byte, the five element will be stored as follows

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
Address 1000	1002	1004	1006	1008

Here variable **arr** will give the base address, which is a constant pointer pointing to the element, **arr[0]**. Therefore **arr** is containing the address of **arr[0]** i.e 1000.

arr *is equal to* **&arr[0]** // by default

We can declare a pointer of type int to point to the array arr.

```
int arr[5]={ 1, 2, 3, 4, 5 };
int *p;
p = arr;
or p = &arr[0]; //both the statements are equivalent.
```

Now we can access every element of array **arr** using **p++** to move from one element to another.

NOTE: You cannot decrement a pointer once incremented. p-- won't work.

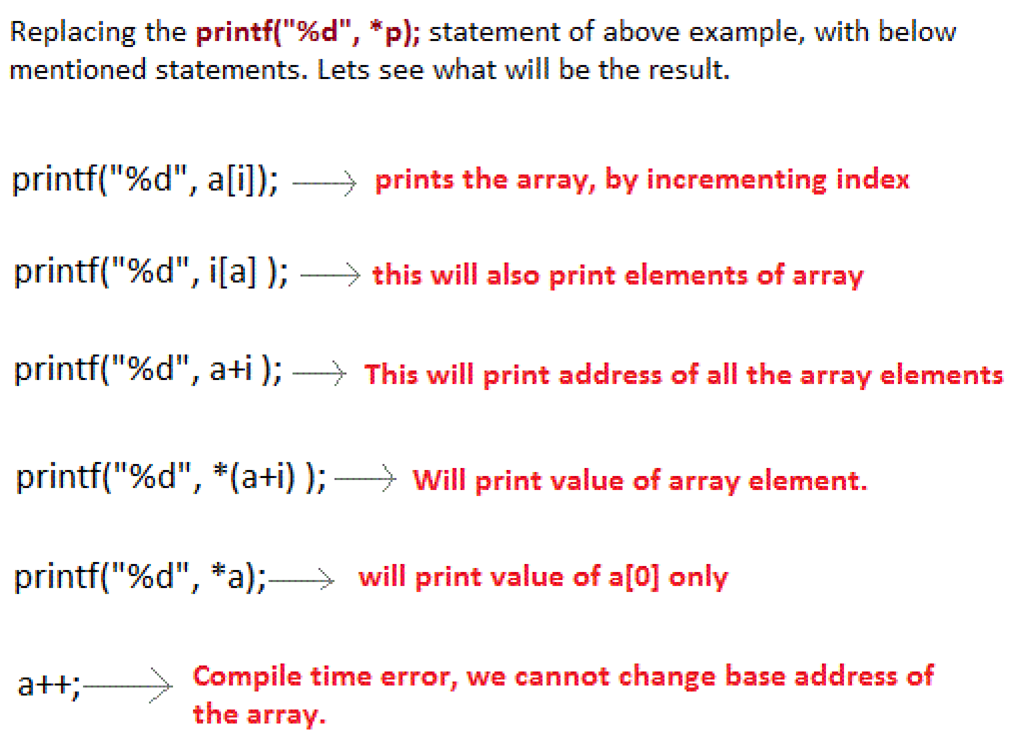
Pointer to Array

We can use a pointer to point to an Array, and then we can use that pointer to access the array. Lets have an example,

```
int i;
int a[5] = {1, 2, 3, 4, 5};
int *p = a; // same as int*p = &a[0]
for (i=0; i<5; i++)
{
printf("%d", *p);
p++;
    }
```

In the above program, the pointer ***p** will print all the values stored in the array one by one

We can also use the Base address (**a** in above case) to act as pointer and print all the values.


Replacing the **printf("%d", *p);** statement of above example, with below mentioned statements. Lets see what will be the result.

printf("%d", a[i]); → prints the array, by incrementing index

printf("%d", i[a]); → this will also print elements of array

printf("%d", a+i); → This will print address of all the array elements

printf("%d", *(a+i)); → Will print value of array element.

printf("%d", *a); → will print value of a[0] only

a++; → Compile time error, we cannot change base address of the array.

Relation between Arrays and Pointers

int arr[5];

arr[0]	**arr[1]**	**arr[2]**	**arr[3]**	**arr[4]**

in this above array is point first position address by default.

&arr[0] it will be print address of 0th location and it same as **arr**

&arr[0] is equivalent to arr

arr[0] will print value of 0th location and ***arr** it will be print value of 0th location.

Similarly,

&arr[1] is equivalent to (arr + 1) AND, arr[1] is equivalent to *(arr + 1).

&arr[2] is equivalent to (arr + 2) AND, arr[2] is equivalent to *(arr + 2).

&arr[3] is equivalent to (arr + 3) AND, arr[3] is equivalent to *(arr + 3).

Pointer Arithmetic and Arrays

Pointer hold/store address of variable so they can be arithmetic operation on the pointer variable .all type of arithmetic operations are not possible with pointer the only valid operations that can be performed

- Increment
- Decrement
- Addition
- Subtraction
- Comparison

Increment pointer:

- If we increment a pointer by 1, the pointer will start pointing to the immediate next location.
- Incrementing Pointer is generally used in array because we have contiguous memory in array and we know the contents of next memory location.
- Incrementing Pointer Variable Depends Upon data type of the Pointer variable.

Rule to increment the pointer

new_address= current_address + i * size_of(data type)

Address + 1 = Address
Address++ = Address

++Address = Address

Note :

32 bit compiler

Prasant Dash
7008191907

For 32 bit int variable, it will increment to 2 byte.

64 bit compiler

For 64 bit int variable, it will increment to 4 byte

Data Type	**Older Address stored in pointer**	**Next Address stored in pointer after incrementing (ptr++)** (32 bit)
int	1000	1002
float	1000	1004
char	1000	1001

Experiments: #01

```
#include <stdio.h>
void main()
{
int a=20;
int *p;//pointer to int
p=&a;//stores the address of avariable
printf("Address of p variable is %u \n",p);
p=p+1;
printf("After increment: Address of p variable is %u \n",p);
}
```

Experiments: #02

```
#include<stdio.h>
void main ()
{
   int arr[5] = {1, 2, 3, 4, 5};
   int *p = arr;
```

```
    int i;
    printf("printing array elements...\n");
    for(i = 0; i< 5; i++)
    {
        printf("%d  ",*(p+i));
    }
}
```

Decrement(--)

Formula of decrementing pointer

new_address= current_address - i * size_of(data type)

Experiments: #03

```
#include <stdio.h>
void main()
{
int a=20;
int *p;//pointer to int
p=&a;//stores the address of avariable
printf("Address of p variable is %u \n",p);
p=p-1;
printf("After increment: Address of p variable is %u \n",p);
}
```

Addition(+)

When a pointer is added with an integer value, the value is first multiplied by the size of the data type and then added to the pointer.

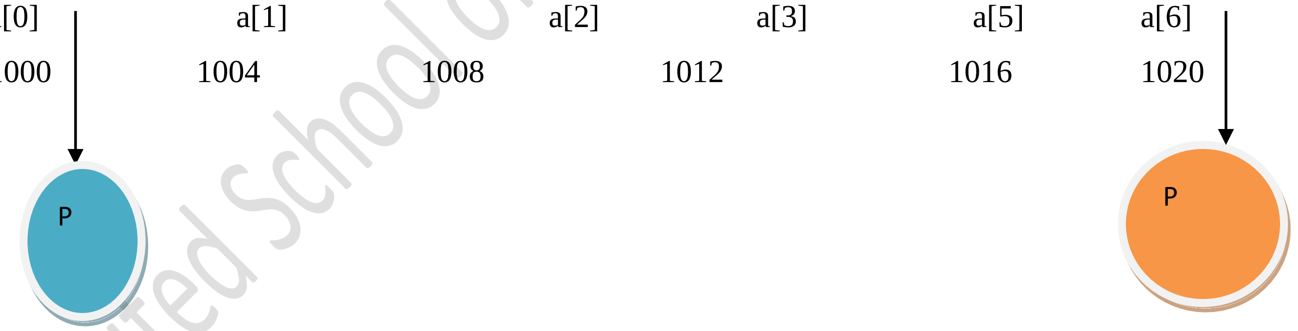
For Example:

ptr is an **integer pointer** that stores **1000** as an address. If we add integer 5 to it using the expression, **ptr = ptr + 5,** then, the final address stored in the ptr will be **ptr = 1000 + sizeof(int) * 5 = 1020**

```
#include <stdio.h>
void main()
{
int a=20;
int *p;//pointer to int
p=&a;//stores the address of a variable
printf("Address of p variable is %u \n",p);
p=p+5;
printf("After increment: Address of p variable is %u \n",p);
}
```

32 bit compiler

10	20	30	50	40	60
a[0]	a[1]	a[2]	a[3]	a[5]	a[6]
1000	1004	1008	1012	1016	1020



Subtraction (-)

When a pointer is subtracted with an integer value, the value is first multiplied by the size of the data type and then subtracted from the pointer similar to addition.

ForExample:
Consider the same example as above where the **ptr** is an **integer pointer** that stores **1000** as an address. If we subtract integer 5 from it using the expression, **ptr = ptr – 5,** then, the final address stored in the ptr will be **ptr = 1000 – sizeof(int) * 5 = 980.**

Comparison of Pointers

We can compare the two pointers by using the comparison operators in C. We can implement this by using all operators in C >, >=, <, <=, ==, !=. It returns true for the valid condition and returns false for the unsatisfied condition.

```
#include <stdio.h>

int main(int argc, char *argv[])
{
        int a=50,*ptr,*p;
        ptr=&a;
        p=&a;
        if (p==ptr)

        printf("address equal \n");
        else
        printf("address not equal \n");

        return 0;
}
```

Passing an Array to a Function

If you want to pass a single-dimension array as an argument in a function, you would have to declare a formal parameter in one of following three ways and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received. Similarly, you can pass multi-dimensional arrays as formal parameters.

1) Formal parameters as a pointer –

```
void myFunction(int *param)
{
.
.
}
```

Prasant Dash
7008191907

2) Formal parameters as a sized array –

```
void myFunction(int param[10])
{
.
.
.
}
```

3) Formal parameters as an unsized array −

```
void myFunction(int param[ ])
{
.
.
.
}
```

Experiments #01 Program to calculate the sum of array elements by passing to a function

```
#include <stdio.h>
float calculateSum(float num[],int no);

int main()
        {
  float result, num[50];
  int no,i;

  printf("enter array size:");
  scanf("%d",&no);
```

```
            for(i=0;i<no;i++)
                {
              printf("enter array elements:");
              scanf("%f",&num[i]);
                }

  result = calculateSum(num,no);
  printf("Result = %.2f", result);
  return 0;
}

float calculateSum(float num[],int no)
            {
  float sum = 0.0;

  for (int i = 0; i < 6; ++i)

    sum += num[i];

  return sum;
}
```

Experiments #02 pass an entire array to a function argument

```
#include <stdio.h>

void arrpass(int *arr,int size)
{
            int i;
            for(i=0;i<size;i++)
            {
                    printf("\nvalues of array[%d]:%d",i,*arr);
                    arr++;
               }
          }

          int main()
          {
               int a[20],i,size;

               printf("Enter array size\n");
               scanf("%d",&size);

               for(i=0;i<size;i++)
               {
                    printf("enter array elements:");
                    scanf("%d",&a[i]);
               }
               arrpass(a,size);
               return 0;
}
```

Also we can pass call by values and call by reference by using function array

Array of Pointers

An array of pointers would be an array that holds memory locations. An array of pointers is an indexed set of variables in which the variables are pointers (a reference to a location in memory).

Syntax:

data_type_name * variable name

Example

```
int *ptr[MAX];
```

Array XYZ[]	**Pointer *a***
XYZ [0]	*a
XYZ [1]	*(a+1)
XYZ [2]	*(a+2)
XYZ [3]	*(a+3)
XYZ [*n*]	*(a+*n*)

Experiments#01

```
#include <stdio.h>
int main()
{
        int a[20],i,size;
        int *ptr[20];
        printf("Enter array size\n");
        scanf("%d",&size);
        for(i=0;i<size;i++)
        {
                printf("enter array elements:");
                scanf("%d",&a[i]);
        }

        for(i=0;i<size;i++)
                ptr[i]=&a[i];

        for(i=0;i<size;i++)
                printf("\nelements of pointer array[%d]:%d",i,*ptr[i]);
        return 0;
}
```

String pointer

```
#include <stdio.h>



int main()
{

            int i,size,c;
  char *ch[4] = {
    "Chennai",
    "Kolkata",
    "Mumbai",
    "New Delhi"
  };
            for(i=0;i<size;i++)
            {
            c = 0;
    while(*(ch[i] + c) != '\0') {
      printf("%c", *(ch[i] + c));
      c++;
    }
    printf("\n");
            }

            return 0;
}
```

Void pointer:

A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be typcasted to any type. **Special type of pointer** called void pointer or **general purpose pointer**

Declaration of void pointer

```
void * pointer_name;
```

Void pointer example

```
void *ptr; // ptr is declared as Void pointer
char cnum;
int inum;
float fnum;
ptr = &cnum; // ptr has address of character data
ptr = &inum; // ptr has address of integer data
        ptr = &fnum; // ptr has address of float data.
```

Void pointers in C are used to implement generic functions in C.

Note: void pointers cannot be dereferencing. For example the following program doesn't compile.

```
#include<stdio.h>
int main()
{
  int a = 10;
void *ptr = &a;
printf("%d", *ptr);
return 0;
}
```

Output:

Compiler Error: 'void*' is not a pointer-to-object type

Overcome this problem

```
#include<stdio.h>
int main()
{
	int a = 50;
void *ptr = &a;
printf("%d", *(int *)ptr);
return 0;
}
```

Output: 50