# Assignment No - 9

Q1) What is pointer? explain the term with an example.

→ Pointer is considered as derived data type in C, C++.

In java their is no such concept of pointer.

- pointer is considered as variable which stores memory address.
- Pointer is a special variable which stores address of anything.
- The power of pointer is to fetch the data whose address is stores in it.
- we can create pointer which points to any primitive data type.
- Consider below statement & statement reading for pointer.

fetching ——→ int *ip = &i

Capacity of pointer

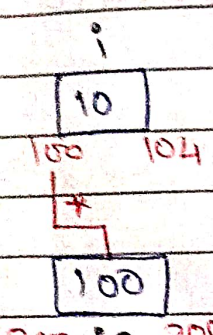
ip is a pointer which points the integer data type currently it holds the address of i.

- Pointer is a variable which holds address, each address is of type unsigned, long due to which size of every pointer is 8 byte.

eg-

```
#include<stdio.h>
int main()
{
  int i = 10;
  int *ip = &i;
  printf("value of i %d\n", i);   //10
```

```
printf("Value pointed by ip %d\n", *ip);   // 100
printf("Address of i %u\n", &i);           // 100
printf("Address of ip %u\n", &ip);         // 200
printf("Address hold by ip %u\n", ip)      // 100
printf("Size of i %d\n", sizeof(i));       // 4
printf("Size of ip %d\n", sizeof(ip));     // 8
return 0;
}
```




output i = 10.
&i = 100
ip = 100
&ip = 200
ip = 10
sizeof(i) = 4byte
sizeof(ip) = 8 bytes

Q2 What is mean by NULL pointer? Why we must initialize the pointer to NULL?

→ - A null pointer in programming refers to a pointer that does not point to any valid memory location or object.
- It is used as a placeholder or sentinel to indicate that the pointer is intentionally empty or invalid.
- A null pointer ensures that the pointer is not accidentally pointing to an arbitrary or unintended memory location.
- It is typically used in error checking, function return values, or to signify the end of data structures.

Initializing a pointer to NULL is important

because it ensures that the pointer does not hold a random or garbage memory address.

~~Uninti~~ Uninitialized pointers can lead to unpredictable program behavior, such as undefined behavior or segmentation faults, when they are dereferenced.

- Reason to initialize pointers to NULL

① Avoid Undefined Behavior
② Safe Dereferencing
③ Error Indication
④ Debugging and code clarity.

Q3] What are different types of pointers? list out them.

→ Pointers in C can be classified into several types based on their usage and the types of data they point.

1] Integer pointer
-these pointer are pronounced as pointer to int
e.g- int *ptr;

2] Array pointer -
A pointer that points to the entire array.
eg- int arr[5] = {1, 2, 3, 4, 5};
int (*ptr)[5] = &arr;

3] Structure pointer -
e.g- struct struct_name *ptr;

4] Function Pointers -
A pointer that points to the address of a function. It can be used to call

functions indirectly.
e.g- int (*ptr) (int, char);
5] Double pointer (pointer to pointer)
A pointer that stores the address of another pointer.
e.g. datatype ** pointer_name;
6] NULL pointer -
The null pointers are those pointer that do not point to any memory location.
e.g. - datatype *pointer_name = NULL;
7] Void pointer -
They are also called generic pointer as they can point to any type and can be typecasted to any type.
e.g. - void *pointer-name;
8] Wild pointer -
An uninitialized pointer that points to an unknown or random memory location.
e.g. int *ptr;
char *str;
9] Constant pointer -
In constant pointer the memory address stored inside the pointer is constant and cannot be modified.
e.g. data-type * const pointer-name;
10] Pointer to constant -
The pointers pointing to a const value that cannot be modified called pointers to constant.
e.g. const data-type *pointer-name;

• Other types of pointers in C:

- Far pointer - A far pointer is typically 32-bit that can access memory outside the current segment.
- Dangling pointer - A pointer pointing to a memory location that has been deleted is called as dangling pointer.
- Huge pointer - A huge pointer is 32-bit long containing segment address and offset address
- Complex pointer - Pointers with multiple levels of indirection
- Near pointer - is used to store 16-bit addresses means within the current segment on a 16-bit machine.
- Normalized pointer - It is a 32-bit pointer, which has as much of its value in the segment register as possible.
- File pointer - The pointer to a file data type is called Stream pointer or file pointer

Q4 Write notes on pointer, size of pointer, data type of pointer.

→

A pointer is a variable that stores the memory address of another variable. Instead of holding a value directly, it "points" to a location in memory where the value is stored.
Pointers are a powerful features in C that allow for dynamic memory management, Function argument passing, and efficient data structure implementation.

• Size of pointer
Every pointer is of 8 byte because it holds the memory address and memory address is of unsign, long.
- If the pointer is char* then it will fetch 1 byte
- If the pointer is int* or float* then its data fetching capacity is 4 byte.
- If pointer is double* fatching capacity is 8 byte.
- * operator is used with the pointer to fetch the data of that pointed data
- * called as dereferencing operator.
• We can create a pointer which points to any primitive data type.
- Data Types of pointer
pointers are strongly associated with the data type they point to. The data type determines how many bytes the pointer dereferences and interprets at the memory address.

Declaration → int *int_ptr;
char *char_ptr;
float *float_ptr;

Data types are important because it determines the size of data and ensures type safety.

Q5] What is mean by increment and decrement operator?
→ In C, C++, Java their are two operators which are used to increment the value as 1 As well as decrement the value by 1.

Increment → (++)
Decreament → (--)

## Short Hand operators

No = 0
i = 10

| Pre increment | Post increment | Pre Decreament | Post Decreament |
|---|---|---|---|
| No = ++i; = first increment then assign | No = i++ first assign then increment | No = --i first decrement then assign | No = i-- first assign then decrement |
| No 0 11 | No 0 10 | No 0 9 | No 0 10 |
| i 10 11 | i 10 11 | i 10 9 | i 10 9 |

All above operators are considered as short hand operators.

- Suppose No is a variable which is off type int and initialized with the value 50.

```
int No = 50
No ++;
```

It is just a increment not pre not post

- Above statement internally converted as

```
No = No + 1;
```

- The same thing get happens with the decrement operator.

```
No --;  // just decrement.
```

Internally converted as No = No-1

Q6] What is pointer to pointer. explain with example and write a program to demonstrate this.

→ A pointer to pointer is a pointer variable that stores the address of another pointer. It is also called a double pointer. This allows for an additional level of indirection where you can manipulate a pointer through another pointer.

- A pointer to a pointer can store the address of another pointer.
- It is declared using two asterisks (**)
  int **ptr;
- Dereferencing once (*ptr) gives the value of the first pointer, and dereferencing twice (**ptr) gives the value stored at the address pointed to by the first pointer.

eg-

```
#include<stdio.h>
int main()
{
  int num = 10;
  int *ptr = &num;
  int **dptr = &ptr;
  printf("Value of num : %d\n", num);
  printf("Address of num: %p\n", &num);
  printf("Value of ptr : %p\n ", ptr);
  printf ("Value pointed to by ptr : %d\n", *ptr);
  printf ("Value of dptr : %p\n", dptr);
  printf ("Value pointed to by dptr: %p\n", *dptr);
  printf ("Value pointed to by the pointer to pointer: %d\n",
                                                **dptr);
  return 0;
}
```

Q 7] Write note on pointer arithmetic, explain in details with example.

→ Generally there are four arithmatic operation
+, −, ×, ÷

- In case of pointer these are not like a normal arithmatic operator that we performe on numeric value.
- According to the arithmatic operation *, ÷ not allow.

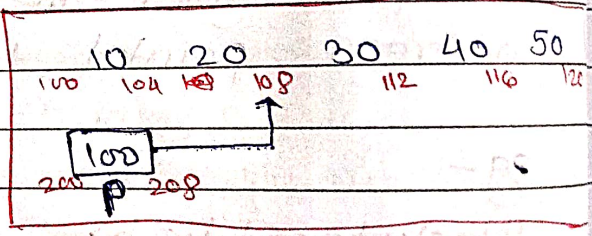
Different Arithmatic operation.

① Addition

- Addition of pointer with number (PTR + NO)

e.g. P+2;

P+2 = P+2 * Sizeof (pointer data type)
= P+2 * Sizeof (int)
= P+2*4
= P+8
= 108




```
int Arr[5] = {10, 20, 30, 40, 50}
int *p = Arr
int *q = &(Arr[4]);
output - 10 20 30 40 50
```

- Addition of two pointer (PTR + PTR)

Not allow

P+q = 100 + 116 = 216

- where 216 is not found in our array

Because,

when we perform addition of two pointer then we get memory address which is having

above that it will not be present our memory if you do so then it is considered as inforcement to other process & too avoid that macroprocess kill that.

2] Substraction

- Substract no from pointer (PTR-NO)

eg. q-2

q-2 = q-2 * sizeof (pointer data type)
= q-2 * sizeof (int)
= q-2 * 4
= q-8
= 116-8
= 108

- Substracting two pointer (PTR-PTR)

q-p = (q-p) / sizeof (pointer data type)
= (116-100) / sizeof (int)
= 16/4
= 4

Now q pointer will store address as a two

3] Increment operator

p++ = (p+1)
= (p+1) * sizeof (pointer data type)
= p+1 * sizeof (int)
= p+1 * 4
= 100+4
= 104

4] Decrement operation

q-- = (q-1)
= (q-1) * sizeof (pointer datatype)
= 116-1* size of int
= 116-1*4
= 116-4
= 112

- To used the concept of pointer arithmatic their should be atleast two pointer.
- Both should be of same data type.
- Both pointer should point to same memory region (expected is array).

Q8] Explain how array is considered as pointer & pointer can be treated as Array.

→

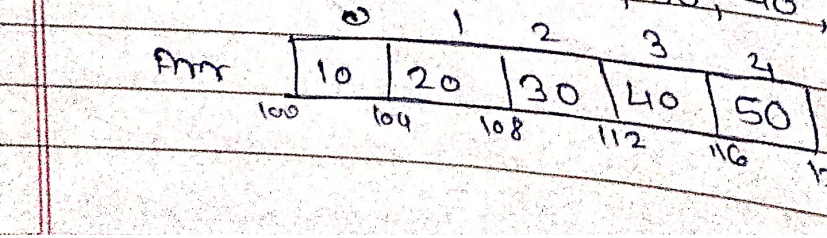
Array is internally considered as a pointer. As a programmer when we access any element of array compiler will internally convert the syntax into its corresponding pointer's representation.

- Concept of array & pointer and its co-relation is same in C & C++ programming.

- ~~Array of const data members~~
~~Const int Arr[4] = {10, 20, 30, 40};~~

Consider below example.

int Arr[5] = {10, 20, 30, 40, 50};



when we considered name of array it will give base address of its 0th element.

printf ("%d\n", Arr); //100

- when we use the & operator with name of Arr it will give address of whole Array.
  i.e. Arr +1
- It get shifted by size of single element. Single element 4 byte because integer data type.

printf("%d\n", Arr+1); // 104

- when we use +1 operation with & operator.
  i.e. & Arr+1,
- Then it will give the address which is shifted by size of whole array.

printf ("%d\n", & Arr+1); //120

- when we access any element of array by using [] operator, it is internally converted to its corresponding pointer representation.

- Internally pointer representation

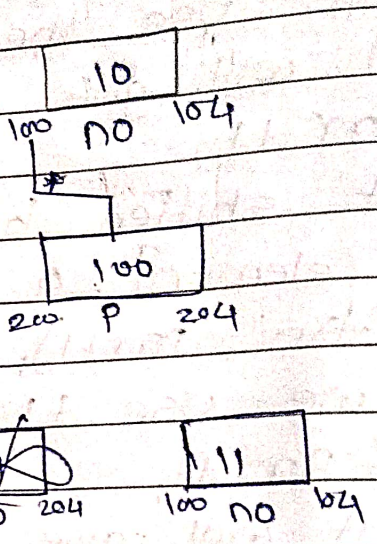
  Arr [3]
  * (Arr+3)
  * (100+3)
  * (100+3(4))
  * (100+12)
  * (112)
  40

In case of array indexing strart from 0 because due to its internal pointer representation if we start array index from 1 internally.

Arr [i] is internally consider as * Arr+i

Q9] predict the output of below code.

```
#include<stdio.h>
int main()
{ int no = 10;
  int *p = NULL;
  p = &no;
  printf("%d", no); //10
  printf("%d", p); //100
  *p = 11;
  printf("%d", no); //11
  printf("%d", p); //100
return 0;
}
```



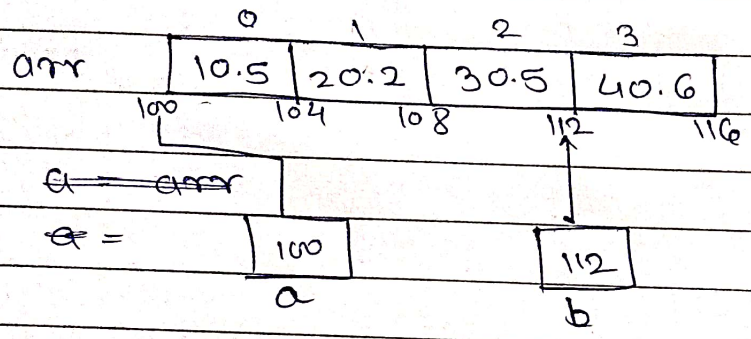

Output → 10100 11100

Q10] Predict the output of below code.

```
#include<stdio.h>
int main()
{ float arr[] = {10.5, 20.2, 30.5, 40.6};
  float *a = NULL;
  float *b = NULL;

  a = arr; //100
  b = &(arr[3]); //112
  printf("%d", a); //100
  printf("%d", b); //112
  printf("%f", *b); //40.6
  printf("%f", *a); //10.5
  printf("%f", *(a+2)); 30.5
```

```
printf ("%f ", *(a+1)); //20.2
printf ("%f", a[1]); //20.2
printf ("%f ", *(2+arr)); //30.5
printf ("%f", 0[arr]); // 10.5
printf ("%f ", b-a); // 10.5
printf ("%f ", *(b-2)); // 20.2
return 0;
}
```




output - 100 112 40.6 10.5 30.5 20.2 20.2 30.5 10.5 10.5 20.2