Pointers

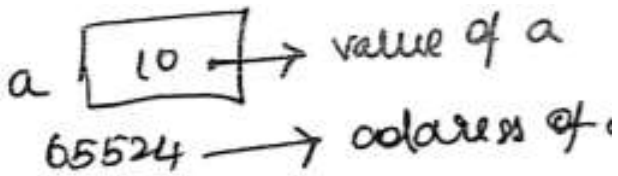
Pointers

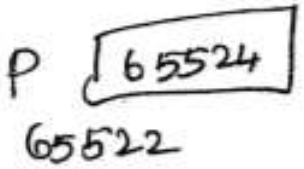
- Pointers in C language is a variable that stores/points the address of another variable.
- A Pointer in C is used to allocate memory dynamically i.e. at run time.
- The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.
- Pointer usage reduces the access time of a variable.
- The size of the pointer depends on the architecture.

```
int a;
a=10;

int *p;

p=&a;
```





Operators used with pointers

1. Referencing operator (&)

   - used to specify address of a variable

2. de-referencing operator (*)

   - used for two purposes,

   a) pointer declarator

   b) value at operator

Syntax

Declaration:

type *ptr_variable;

Example:

int *p;

Initialization:

int n = 10; // variable whose address need to be stored into the pointer

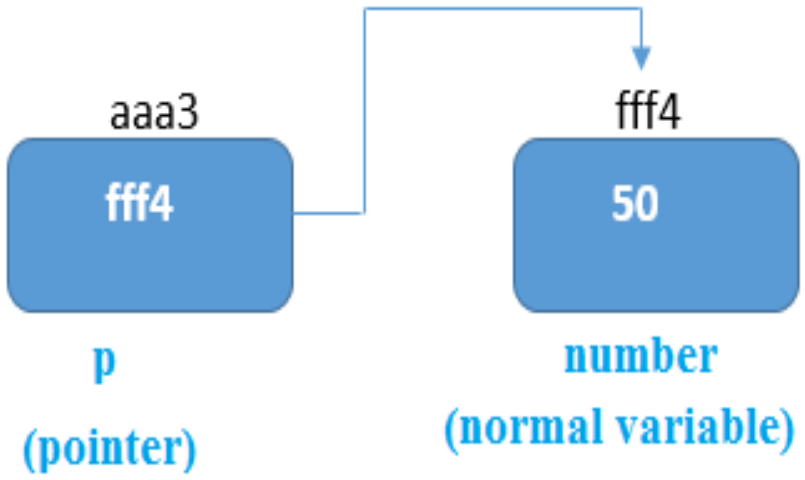
int* p; // declaring pointer variable

p=&n;// storing address of ‘n’ into the pointer variable ‘p’

Pointer Address Format

Pointer Example

An example of using pointers to print the address and value is given below.



Explanation

- 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.

Address Of (&) Operator

- The address of operator '&' returns the address of a variable. But, we need to use %u or %p format specifier to display the address of a variable.

```
#include<stdio.h>
int main(){
int number=50;
printf("value of number is %d, number);
printf(“ address of number is %p“, &number);
return 0;
}
```

Example program

```
#include <stdio.h>

int main()
{
int a=10;
int *p;
p=&a;
printf("\n a: %d",a);
printf("\n &a: %u",&a);
printf("\n p: %u",p);
printf("\n p: %u",&p);
printf("\n *p: %d",*p);
}
```

Example program

```
#include <stdio.h>

int main()
{
int a=10;
int *p;
p=&a;
printf("\n a: %d",a);
printf("\n &a: %u",&a);
printf("\n p: %u",p);
printf("\n p: %u",&p);
printf("\n *p: %d",*p);
}
```

Output:

```
   a: 10
 &a: 1386055356
   p: 1386055356
    p: 1386055344
  *p: 10
```

Size of pointer

- On 64-bit machines, pointers take up 8 bytes of memory
- on 32-bit machines, they take up 4 bytes.

```
int main()
{
int *p1;
float *p2;
char *p3;
printf("%d %d %d", sizeof(p1), sizeof(p2), sizeof(p3));
 return 0;
}
```

Answer : 8 8 8

Irrespective of type, since pointers stores only address.

Advantages of Pointers

- Pointer **reduces the code** and **improves the performance**, it is used to retrieve strings, trees, etc. and used with arrays, structures, and functions.
- We can **return multiple values from a function** using the pointer.
- It makes you able to **access any memory location** in the computer's memory.
- Pointers in c language are widely used in arrays, functions, and structures. It reduces the code and improves the performance
- Dynamic memory allocation: In c language, we can dynamically allocate memory using malloc() and calloc() functions where the pointer is used.

Add two numbers using Pointers

```
#include <stdio.h>
 int main()
{
   int first, second, *p, *q, sum;
   printf("Enter two integers to add\n");
   scanf("%d%d", &first, &second);
   p = &first;
   q = &second;
   sum = *p + *q;
   printf("Sum of the numbers = %d\n", sum);
   return 0;
}
```

Swapping two numbers using pointers

```
#include <stdio.h>
int main()
{
   int x, y, *a, *b, temp;
   printf("Enter the value of x and y\n");
   scanf("%d%d", &x, &y);
   a = &x;
   b = &y;
   temp = *a;
  *a   = *b;
  *b   = temp;
   printf("After Swapping\nx = %d\ny = %d\n", x, y);
   return 0;
}
```

NULL Pointer

- A pointer that is not assigned any value but NULL is known as the NULL pointer.
- If you don't have any address to be specified in the pointer at the time of declaration, you can assign NULL value.
- Syntax:

```
int *p=NULL;
```

```
#include <stdio.h>
int main () {
int  *ptr = NULL;
   printf("The value of ptr is : %u\n", ptr  );
   return 0;
}
```

Output: The value of ptr is : 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.

```
int a = 10;
char b = 'x';

void*p;
p = &a;  // void pointer holds address of int 'a'
p = &b; // void pointer holds address of char 'b'
```

- When void pointer stores all the types of variable's address, while printing value through pointers it has to be converted to corresponding type before it prints.

```
int a=10;
void *p;
p=&a;
printf("%d", *p);   //statement will not work.
printf("%d", *(int *)p);   // is correct statement.
```

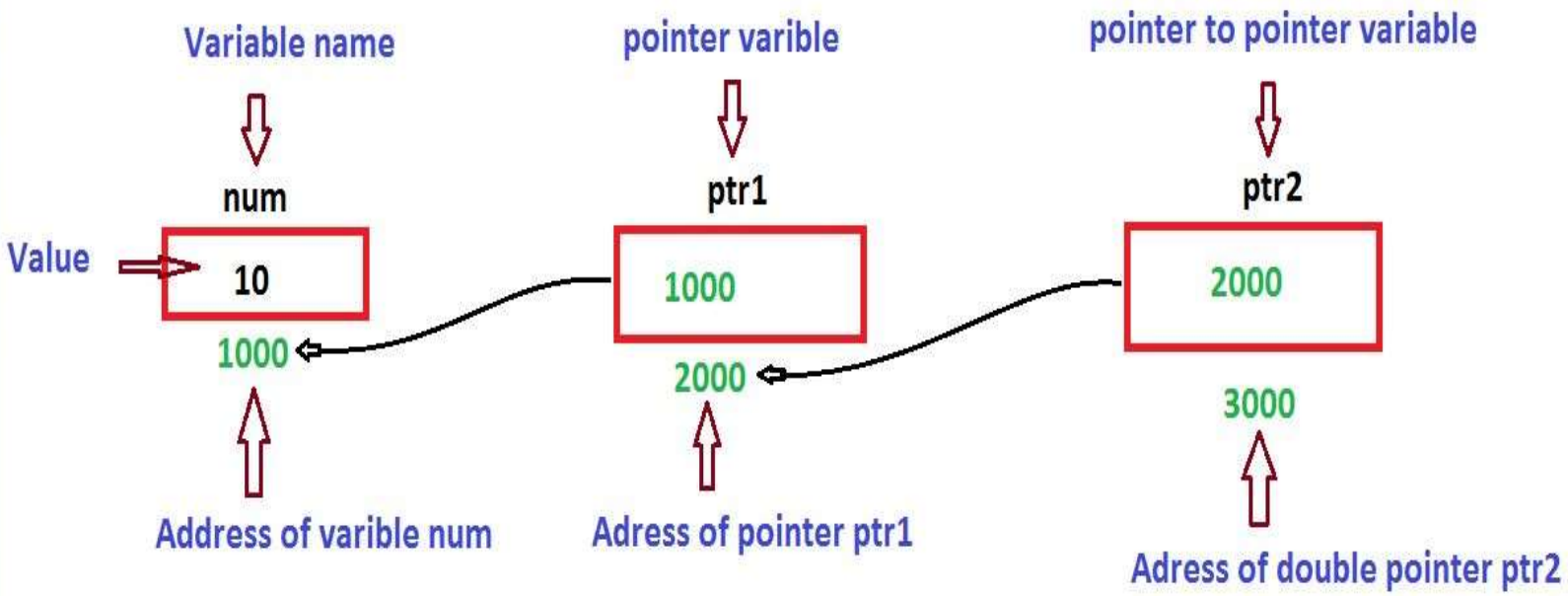
C Double Pointer (Pointer to Pointer)

- pointer to store the address of another pointer.
- A pointer to a pointer is a form of multiple indirection, or a chain of pointers.
- Normally, a pointer contains the address of a variable.
- When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value
- Syntax:
- int **p;

Double Pointer

```
int main () {
   int  num;
   int  *ptr1;
   int  **ptr2;
   num = 10;
   ptr1 = &num;
   ptr2 = &ptr1;
   printf("Value of num = %d\n", num );
   printf("Value available at *ptr1 = %d\n", *ptr 1);
   printf("Value available at **ptr2 = %d\n", **ptr2);
   return 0;
}
```

Double Pointer



Pointer and Arrays

- **Pointer to One dimensional arrays**
- Use a pointer to an array, and then use that pointer to access the array elements.

- **a[i] can be represented as *(a+i)**

Example to access one dimensional array using pointers.

```
#include<stdio.h>

int main()
{
   int a[3] = {1, 2, 3};
   int *p = a;
   for (int i = 0; i < 3; i++)
   {
      printf("%d ", *p);
      p++;
   }
  return 0;
}
```

Output :
1 2 3

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.**

```c
/* Sorting N numbers using pointers – NOSRTPTR.C */

#include <stdio.h>

int main()
{
        int *a, n, i, j, t;
        printf("Enter the number of elements : ");
        scanf("%d",&n);
        printf("Enter the elements : \n");
        for(i = 0; i < n; i++)
                scanf("%d", a + i);
        for(i = 0 ; i < n - 1 ; i++)
                for(j = i + 1; j < n; j++)
                        if(*(a + i) > *(a + j))
                        {
                                t = *(a + i);
                                *(a + i) = *(a + j);
                                *(a + j) = t;

                        }
        printf("The elements in ascending order :\n");
        for(i = 0 ; i < n ; i++)
                printf("%d\t", *(a + i));
}
```

Output

Run 1:

Enter the number of elements : 5

Enter the elements :

20

40

50

30

10

The elements in ascending order :

10 20 30 40 50

```c
/* To compute the sum of all elements in an array – ARSUMPTR.C */

#include <stdio.h>

int main()
{
        int a[10], n, i, sum = 0;
        int *ptr;
        printf("Enter the number of elements :");
        scanf("%d", &n);
        printf("Enter the elements :\n");
        for(i = 0; i < n; i++)
                scanf("%d", &a[i]);
        ptr = a;
        for(i = 0; i < n; i++)
        {
                sum = sum + *ptr;
                ptr++;
        }
        printf("The sum of elements is : %d", sum);
        return 0;
}
```

Output

Run 1:

Enter the number of elements :5

Enter the elements :

10

20

30

40

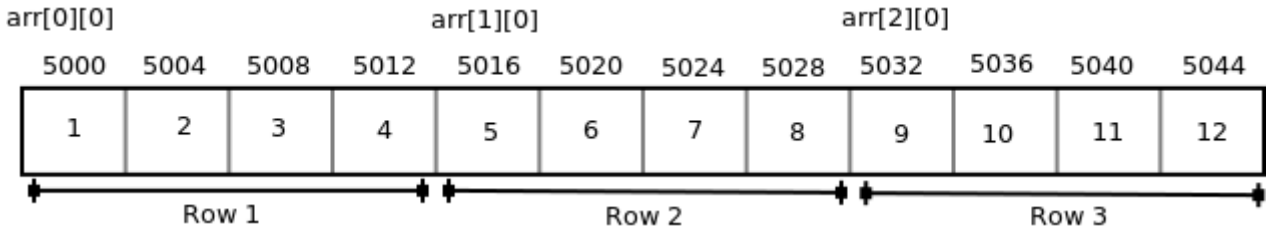
50

The sum of elements is : 150

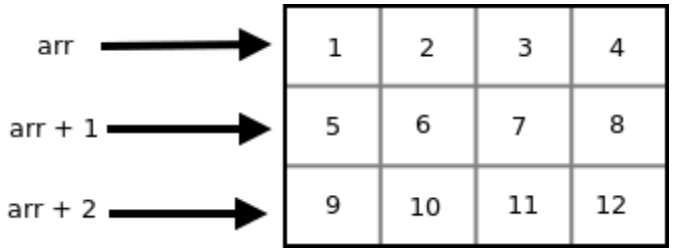
Pointer to multi dimensional arrays

arr[i][j] can be represented as *(*(arr+ i) + j)

- int arr[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };



	Col 1	Col 2	Col 3	Col 4
Row 1	1	2	3	4
Row 2	5	6	7	8
Row 3	9	10	11	12



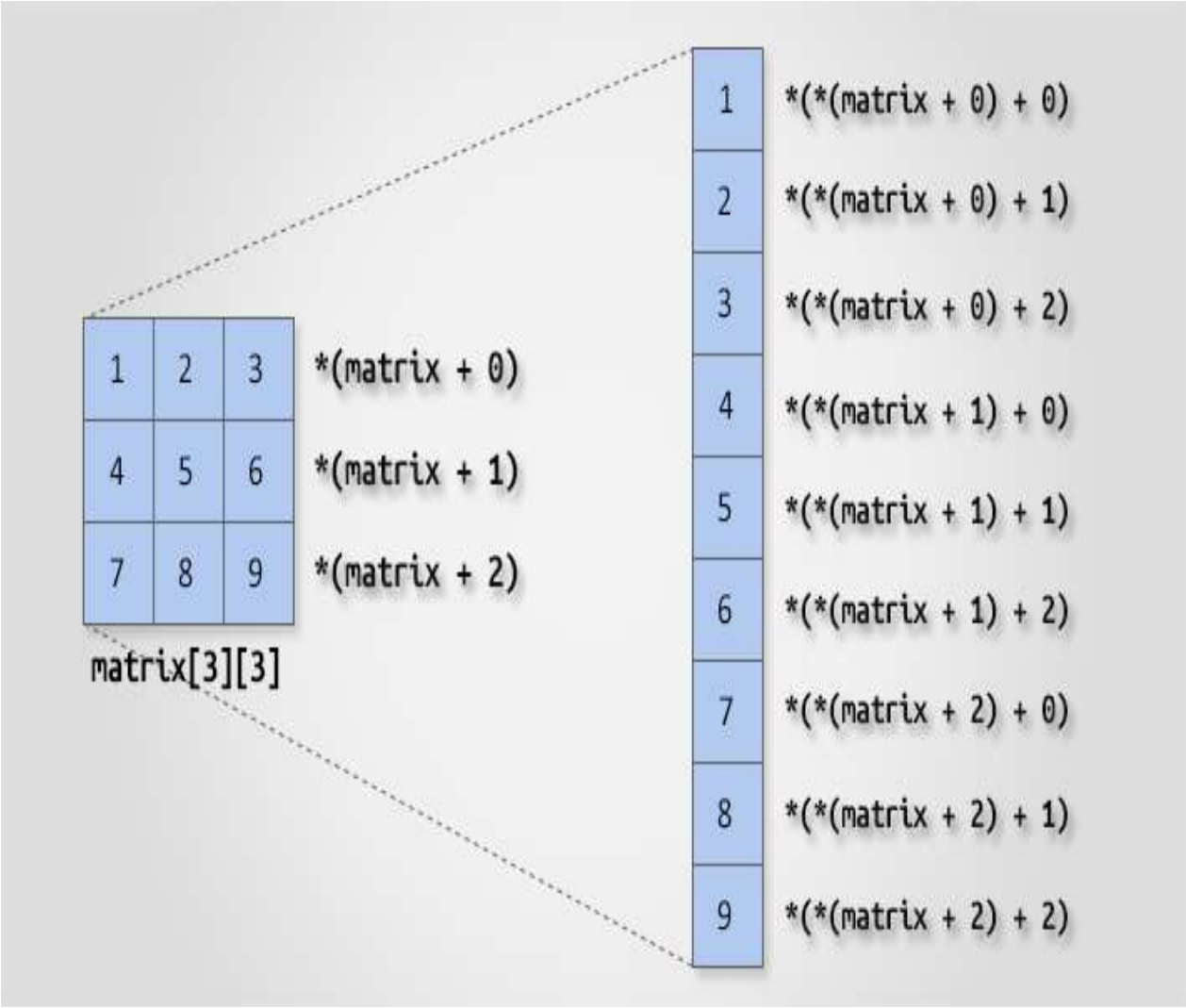
arr - Points to 0th element of arr - Points to 0th 1-D array - 5000
arr + 1 - Points to 1th element of arr - Points to 1nd 1-D array - 5016
arr + 2 - Points to 2th element of arr - Points to 2nd 1-D array - 5032

```
// C program to print the values of a 2-D array
#include<stdio.h>
int main()
{
int arr[3][4] = {{ 10, 11, 12, 13 },
                 { 20, 21, 22, 23 },
                 { 30, 31, 32, 33 }
               };
int i, j;
for (i = 0; i < 3; i++)
{
  for (j = 0; j < 4; j++)
   {
    printf("%d “, *(*(arr + i) + j));
   }
  printf("\n");
}
return 0;
}
```

Output:

```
10 11 12 13
20 21 22 23
30 31 32 33
```

Example : int matrix[3][3];



MCQS

1. A pointer is

A. A keyword used to create variables
B. A variable that stores address of an instruction
C. A variable that stores address of a programmable entity
D. All of the above

1. A pointer is

A. A keyword used to create variables

B. A variable that stores address of an instruction

C. A variable that stores address of a programmable entity

D. All of the above

Answer : C

2. Choose the best one prior to using a pointer variable

A. It should be declared.

B. It should be initialized.

C. It should be both declared and initialized.

D. None of the above.

2. Choose the best one prior to using a pointer variable

A. It should be declared.

B. It should be initialized.

C. It should be both declared and initialized.

D. None of the above.

Answer : C

3. What is the output of this C code?

```
int main()
{
int *ptr, a = 10;
ptr = &a;
*ptr += 1;
printf("%d,%d/n", *ptr, a);
}
```

A. 10,10 B. 10,11 C. 11,10 D. 11,11

3. What is the output of this C code?

```
int main()
{
int *ptr, a = 10;
ptr = &a;
*ptr += 1;
printf("%d,%d/n", *ptr, a);
}
```

A. 10,10 B. 10,11 C. 11,10 D. 11,11

Answer: Option D

4. Find the output.

```
main()
{
char *p;
p="hello";
printf("%c",*&*p);
}
```

A. Hello B. h C. Some address will be printed. D. None of these

4. Find the output.

```
main()
{
char *p;
p="hello";
printf("%c",*&*p);
}
```

A. Hello B. h C. Some address will be printed. D.None of these

Answer : B

5. Comment on the following?

const int *ptr;

A. You cannot change the value pointed by ptr

B. You cannot change the pointer ptr itself

C. Both (a) and (b)

D. You can change the pointer as well as the value pointed by it

5. Comment on the following?

const int *ptr;

A. You cannot change the value pointed by ptr

B. You cannot change the pointer ptr itself

C. Both (a) and (b)

D. You can change the pointer as well as the value pointed by it

Answer: Option A