Pointer
Cloud IT Online batch
Full Lecture

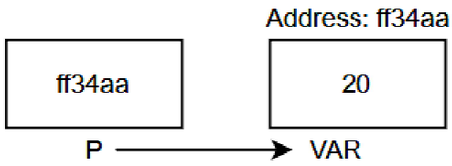
See the following output:

```
#include <stdio.h>
int main()
{
    int *ptr, q;
    q = 50;
    /* address of q is assigned to ptr */
    ptr = &q;
    /* display q's value using ptr variable */
    printf("%d", *ptr);
    return 0;
}
```

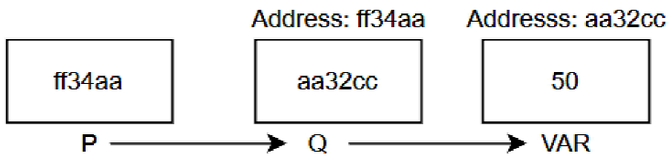
Output: 50

***p vs **p vs ***p meaning:**

***p:** *p is a pointer to a variable, as shown below. It is also called single pointer. The single pointer has two purposes: to create an array and to allow a function to change its contents (pass by reference).



****p:** **p is a pointer to a pointer variable, also called double pointer. It is a form of multiple indirection, or a chain of pointers. 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 as shown below.



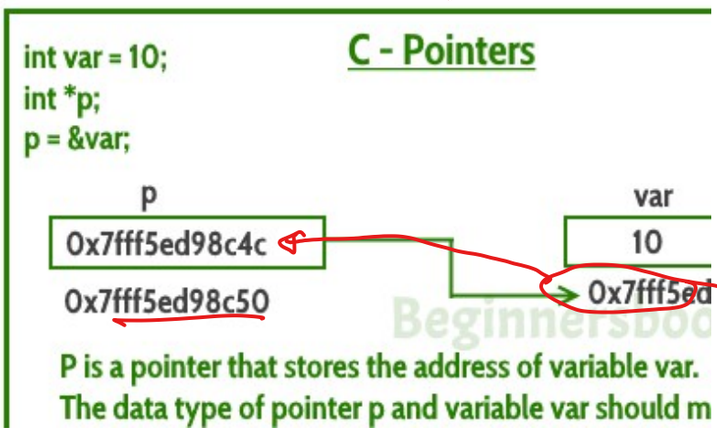
*****p:** ***p is a pointer to a double pointer, rather a pointer to a pointer to a pointer variable, as shown below. It is mostly called triple pointer. It is an even higher level of multiple indirection or pointer chaining. A triple pointer is used to traverse an array of pointers.

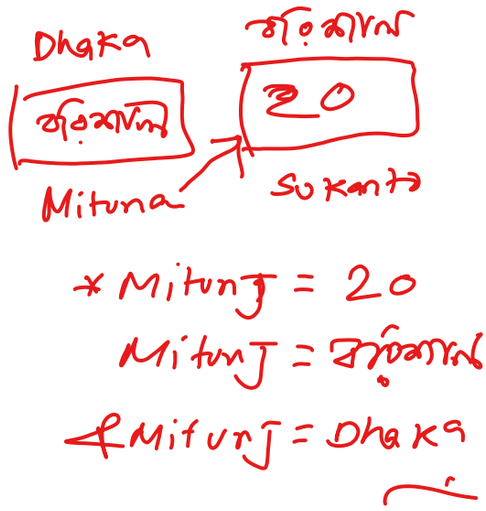
Example of Pointer demonstrating the use of & and *

```
int main()
{
  /* Pointer of integer type, this can hold the
   * address of a integer type variable.
   */
  int *p;

  int var = 10;

  /* Assigning the address of variable var to the pointer
   * p. The p can hold the address of var because var is
   * an integer type variable.
   */
  p= &var;
  printf("Value of variable var is: %d", var);
  printf("\nValue of variable var is: %d", *p);
  printf("\nAddress of variable var is: %p", &var);
  printf("\nAddress of variable var is: %p", p);
  printf("\nAddress of pointer p is: %p", &p);
  return 0;
}
```

Output:

Value of variable var is: 10
Value of variable var is: 10
Address of variable var is: 0x7fff5ed98c4c
Address of variable var is: 0x7fff5ed98c4c
Address of pointer p is: 0x7fff5ed98c50





See the following Example:

```
int main()
{
  int var;
  int *p,**pp,***ppp;
  var =100;
  p=&var;
  pp=&p;
  ppp=&pp;
  printf("value at var= %d\n",var);
  printf("value available at *p= %d\n",*p);
  printf("value available at **p= %d\n",**pp);
  printf("value available at ***p= %d\n",***ppp);
  return 0;
}
```

Output:

value at var= 100
value available at *p= 100
value available at **p= 100
value available at ***p= 100

C Pointer and Array

```
int main()
{
   int array[100]={10,20,30,40};
   printf("%p   ", array);
   printf("%p   ", &array[0]);
   return 0;
}
```

OUTPUT: 0x7ffe66a73930 0x7ffe66a73930. Both address are same. The **memory address** of the **first element** is the same as the **name of the array**:

Since array is a pointer to the first element in array, you can use the * operator to access it:

```
int array[4] = {25, 50, 75, 100};
printf("%d", *array);
```

OUTPUT: 25

To access the rest of the elements in array, you can increment the pointer/array (+1, +2, etc). like `*(array + 1), *(array + 2)`

Understand Pointer how it works:

```
int main () {
int  var = 20;   /* actual variable declaration */
int  *ip;        /* pointer variable declaration */
ip = &var; /* store address of var in pointer variable*/
printf("Address of var variable: %x\n", &var  );
   /* address stored in pointer variable */
printf("Address stored in ip variable: %x\n", ip );
   /* access the value using the pointer */
printf("Value of *ip variable: %d\n", *ip );
return 0;
}
```

Output: Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20

Pointer to Pointer

We know, pointer is a variable that contains address of another variable. Now this variable address might be stored in another pointer. Thus, we now have a pointer that contains address of another pointer, known as pointer to pointer.

Example 3:

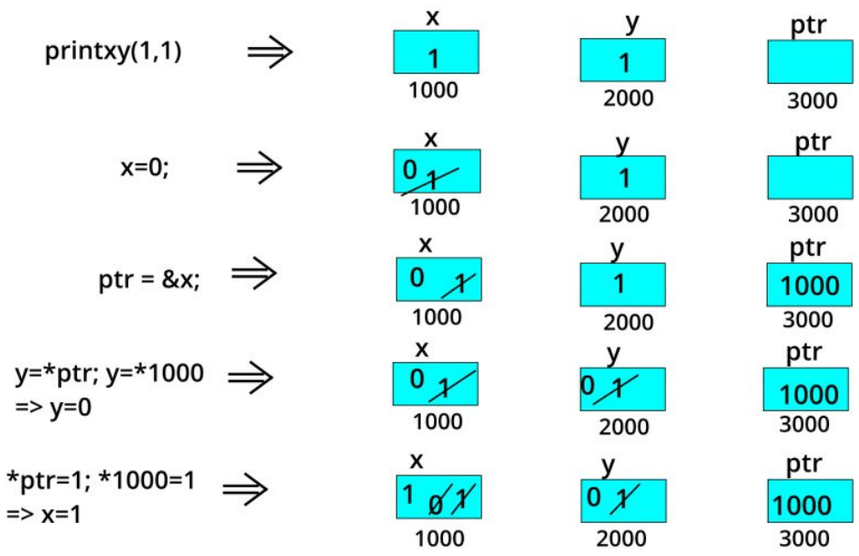
```
main ()
{
int i = 3, *p, **q;
p = &i;
q=&p;
printf("\n Address of i = %u", &i);
printf("\n Address of i = %u", p);
printf("\n Address of i = %u", *q);
printf("\n Address of p= %u", &p);
printf("\n Address of p= %u", q);
printf("\n Address of q = %u", &q);
printf("\n value of i= %d", i);
printf("\n value of i= %d", *(&i));
printf("\n value of i= %d", *p);
printf("\n value of i= %d", **q);
}
```

If the memory map is **q=20555 , *p=20122 and i=2000

Then the **output** is:
Address of i = 2000
Address of i = 2000
Address of i = 2000
Address of p= 20122
Address of p = 20122
Address of q = 20555
Value of i = 3
Value of i=3
Value of i=3
Value of i=3

Example 1:

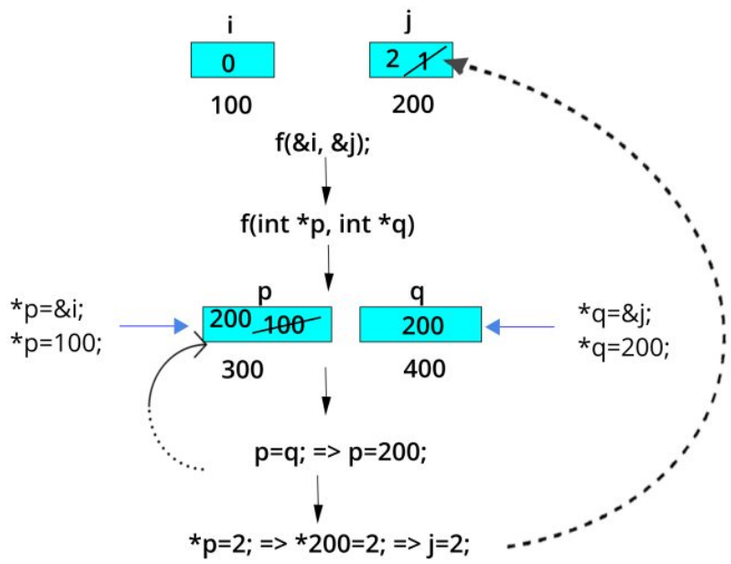
```
void printxy(int x, int y)
{ int *ptr;
 x=0;
ptr=&x;
y=*ptr;
*ptr=1;
printf("%d",%d",x,y);
}
```



printf(%d, %d",x,y); => Correct answer is: 1, 0

Example2:

```
#include<stdio.h>
void f(int *p, int*q)
 {
p=q;
*p=2;
}

int i=0, j=1;
 int main()
{
f(&i, &j);
 printf("%d%d\n",i,j);
 return 0;
 }
```



printf("%d %d\n",i,j); => Correct answer is: 0 2

NULL Pointers

It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.

The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program −

```
int main ()
{
int *ptr = NULL;
printf ("The value of ptr is : %x\n", ptr);
return 0;
}
```

Output: The value of ptr is 0

See the following example

```
int main()
{
 int val[3] = { 5, 10, 20 };
 int *ptr;
 ptr = val ; //assigning all array value to ptr
 printf("Elements of the array are: ");
 printf("%d %d %d ", ptr[0], ptr[1], ptr[2]);
 ++*ptr; // increment the first value of ptr
 printf(" %d %d %d ", ptr[0], ptr[1], ptr[2]);
 printf("  %d",*(++ptr)); // inctrement the index
 return 0;
 }
```

Output: 5 10 20 6 10 20 10

Here 'val' array is assign to ptr. Then ptr point to the array. Print 5 10 20 . Now, ++*ptr means increment the value of ptr *ptr is the first value of array which is 5 ,So,++*ptr=++5=6. So, next output is 6 10 20. And the final line ++ptr means increment the index. Now ptr goes to index 1 which is 10. Print 10.

In most of the operating systems, programs are not permitted to access memory at address 0 because that memory is reserved by the operating system. However, the memory address 0 has special significance; it signals that the pointer is not intended to point to an accessible memory location. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.

- ✓ if(ptr) /* succeeds if p is not null */
- ✓ if(!ptr) /* succeeds if p is null */

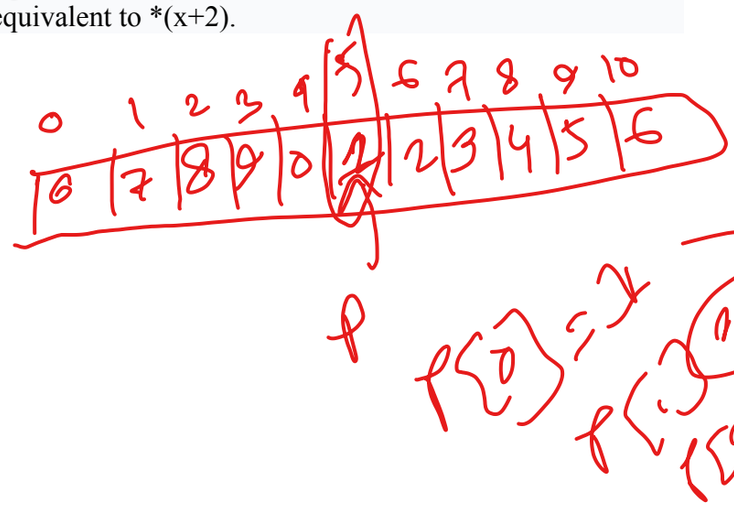
✍ **What is the correct output of the following C program?** *[Com 6 bank AP-2021]*

```
int array[] = {6,7,8,9,0,1,2,3,4,5,6};
*p=array+5;
printf("%d\n", p[1]);
```

a) 1 b) 2 c) 3 d) Compile Error

** it is clear that &x[0] is equivalent to x. And, x[0] is equivalent to *x.
Similarly,

- &x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).
- &x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2).



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**

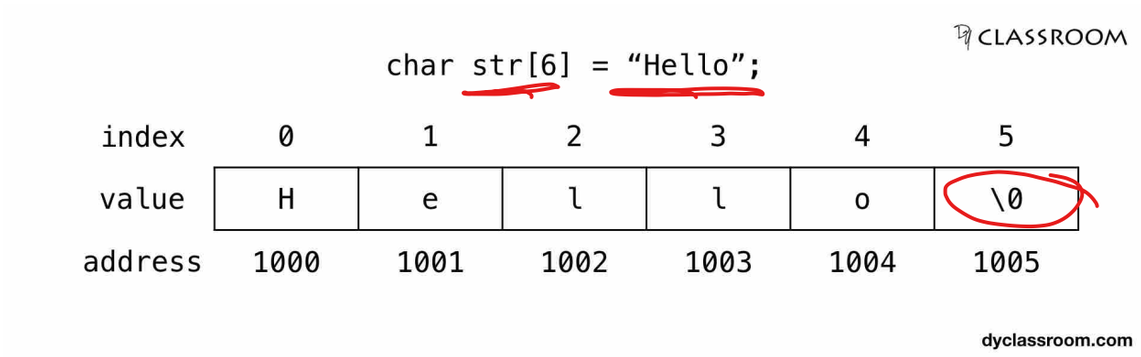
a++;⟶ **Compile time error, we cannot change base address of the array.**

Pointer To String

We know that a string is a sequence of characters which we save in an array. And in C programming language the \0 null character marks the end of a string.

Creating a string

In the following example we are creating a string str using char character array of size 6.

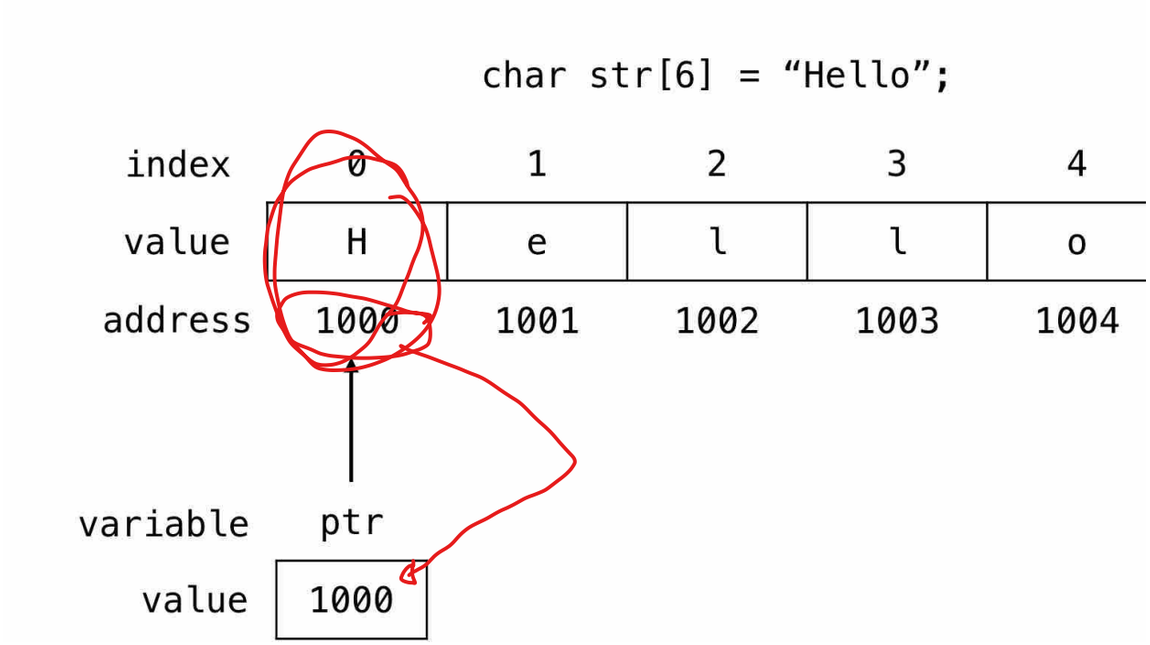


Creating a pointer for the string

The variable name of the string str holds the address of the first element of the array i.e., it points at the starting memory address.

So, we can create a character pointer ptr and store the address of the string str variable in it. This way, ptr will point at the string str.

```
char *ptr = str;
```

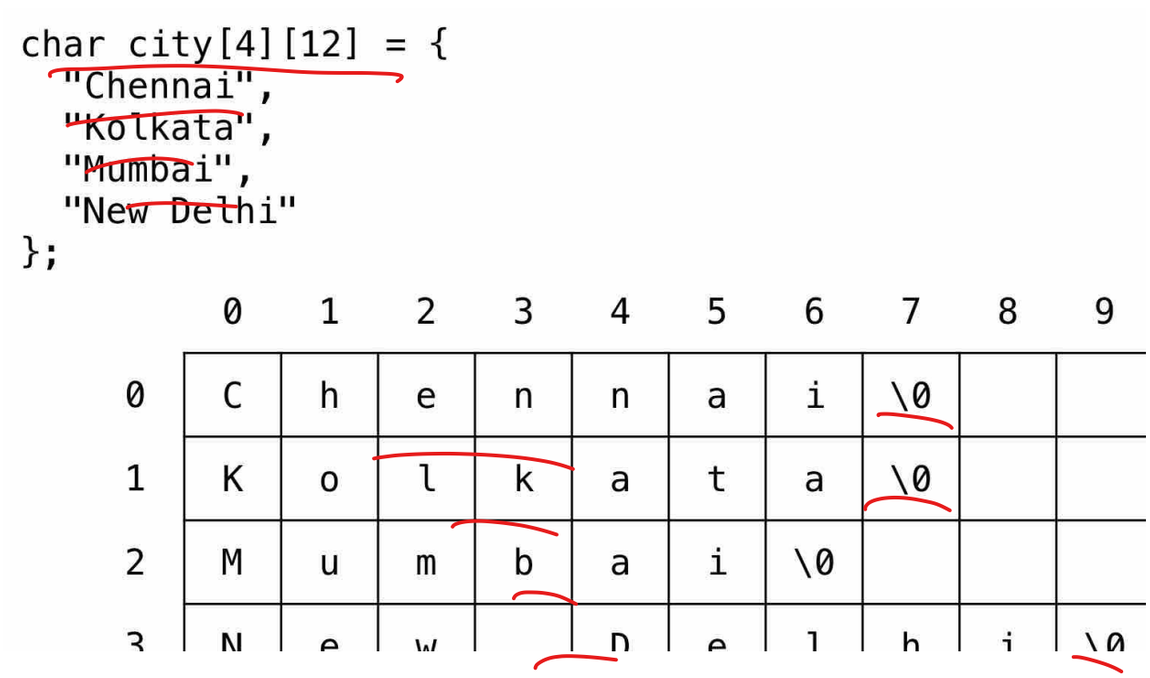


Array of strings

We can create a two dimensional array and save multiple strings in it.

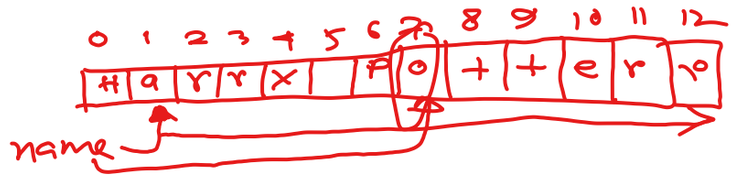
For example, in the given code we are storing 4 cities name in a string array city.

```
char city[4][12] = {
  "Chennai",
  "Kolkata",
  "Mumbai",
  "New Delhi"
};
```



Example of String and array of character:

```
int main(void) {
```



```
char name[] = "Harry Potter";
printf("%s", name);    // Output: Harry Potter
printf("%s", name+1); // Output: arry Potter
printf("%c", *name); // Output: H
printf("%c", *(name+7));    // Output: o
char *namePtr;
namePtr = name;
printf("%c", *namePtr);      // Output: H
printf("%c", *(namePtr+1));    // Output: a
printf("%c", *(namePtr+7));    // Output: o
```

Practices Problem: 1

```
char c[]="GATE2011";
char *p=c;
printf("%s",p+p[3]-p[1]);
```

Ans:

// p[3] is 'E' and p[1] is 'A'.
// p[3] - p[1] = ASCII value of 'E' - ASCII value of 'A' = 4
// So the expression p + p[3] - p[1] becomes p + 4 which is
// base address of string "2011"
Or let the address of p is 2000 so, 2000+E-A=2000+4=2004= p[4]=2

Previous year question:

1. **Which of the following is correct to initial array in C?** *[Com. 6 bank-Ap-2021]*
   a) int array = (1,2,3,4,5) b) int array() = {1,2,3,4,5}
   c) int array() = (1,2,3,4,5) d) int array[5]={1,2,3,4,5}
2. **What is the access methodology in arrays?** *[Com. 6 bank-Ap-2021]*
   a) Sequential b) Random c) Relational d) Stochastic
3. **What is the output of the following program:** *[Competition commision(P)-2019]*

```
int main ()
{
char *a[2] = { "hello", "hi" };
printf ("%s", *(a + 1));
return 0;
}
```

Output: hi

4. **What is the output of following code:***[Competition commision(P)-2019]*

```
int main ()
{
char s[32] = "niksat";
char t[32] = "";
strrev (s);  //taskin
strcpy (t, s);
strcat (t, " so so ");
puts (t);
printf ("%d\n", strcmp ("taskvar", t));
return 0;
}
```

Output: taskin so so

Exercise

```
# include <stdio.h>
void fun(int *ptr)
{
        *ptr = 30;
}

int main()
{
int y = 20;
fun(&y);
printf("%d", y);

return 0;
}
```

Answer: (B)

Explanation: The function fun() expects a pointer ptr to an integer (or an address of an integer). It modifies the value at the address ptr. The dereference operator * is used to access the value at an address. In the statement '*ptr = 30', value at address ptr is changed to 30. The address operator & is used to get the address of a variable of any data type. In the function call statement 'fun(&y)', address of y is passed so that y can be modified using its address.

```
#include <stdio.h>

int main()
{
        int *ptr;
        int x;

        ptr = &x;
        *ptr = 0;

        printf(" x = %d\n", x);
        printf(" *ptr = %d\n", *ptr);

        *ptr += 5;
        printf(" x = %d\n", x);
        printf(" *ptr = %d\n", *ptr);

        (*ptr)++;
        printf(" x = %d\n", x);
        printf(" *ptr = %d\n", *ptr);

        return 0;
}
```