ARRAYS [LEVEL-2]

Page No. _____
Date 08 09 23

Pass By Value :- Whenever a variable is passed into a function by value, it means copy of that passed variable is created in the memory for that particular function.

Ex:-

```
int main(){
    int a=5;
    solve(a);
    cout<<a<<endl;
}
```

```
void solve(int a){
    a++;
    cout<<a<<endl;
}
```

104 a [5]

328 a [56]

Output :-

6
5

↳ Inside main, variable named 'a' is created and after passing 'a' into 'solve' function. Copy of 'a' is created. Here, both variables have same name 'a', but they both are different memory allocations. Thats why output is different bcz scope of 'a' present in main function is limid to main only and 'a' present in solve() function have its scope only inside solve().

Ex:-

```
int main(){
    int mark=90;
    mark++;
    solve(mark);
    cout<<mark<<endl;
    return 0;
}
```

```
void solve(int m){
    m--;
    m=m*10;
    cout<<m;
}
```

104 mark [90] [91]

423 m [91] [90]

Output :-

900 → (m=m×10 ⇒ 90×10 = 900)
91

Page No.
Date

↳ Here, copy of mark is created inside solve() with variable name 'm'. Both the variables have different memory blocks and have different addresses.

Pass By Reference :-

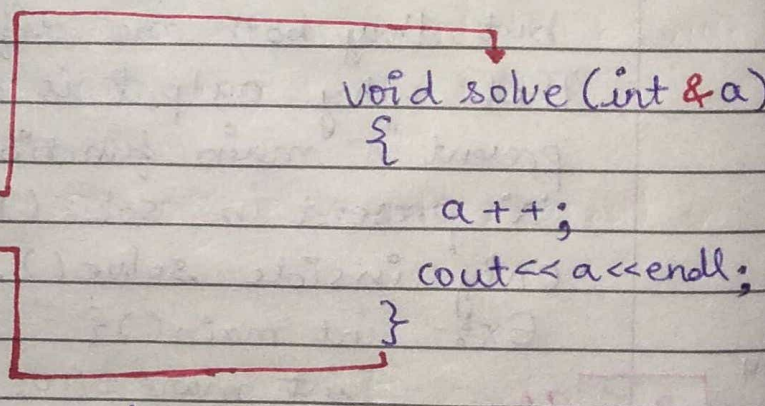
→ When a variable is passed by reference, it means we are sending the original variable into the function and further updation of that variable into that function will also change the value of that variable into the memory block of that variable.

→ In short, further updation of passed variable will gets reflected into its original memory block too.

→ To pass any variable by reference, we write an ampersand '&' sign before the variable name while passing the variable in argument list of particular function.

Ex:-
```
int main(){
    int a=5;
    solve(a);
    cout<<a;
    return 0;
}

void solve(int &a)
{
    a++;
    cout<<a<<endl;
}
```



Output :-

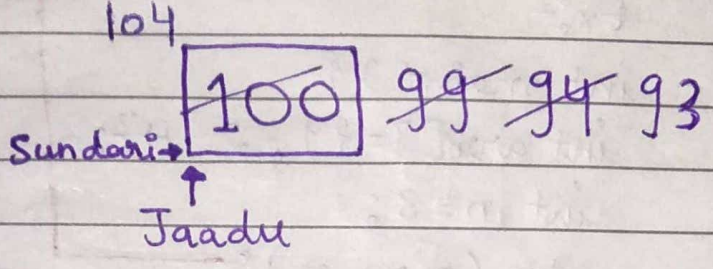
6 → printed by cout written in solve

6 → printed by cout written in main

Page No.
Date

Ex:-
```
int main(){
    int sundari = 100;
    sundari--;
    sundari -= 5;
    solve(sundari);
    cout << sundari;
    return 0;
}
```

```
void solve(int & Jaadu)
{
    Jaadu--;
    cout << Jaddu + 10;
    return;
}
```



Output:-
103
9-

Reference Variable:-

↳ In the above example, variable name 'sundari' and 'Jaadu' both are pointing to same memory block.

↳ Compiler will create a symbol table where both of them are mapped to a same memory address.

Ex:-
```
int a = 5;
int &b = a;
```

∴ let address of a = 1743@2

∴ We can say that, same memory block

Variable Name	Memory Address
Sundari →	104
Jaadu →	104
a →	1743@2
b →	1743@2

Symbol Table

Passing arrays in a function :-

↳ **Keep in Mind** that, array is by default passed by reference. That means original array is passed as an argument every time and any change will get reftect back in its continuous memory blocks.

Ex:-

```
int main(){
 int arr[]={1, 2, 4};
 int n=3;
 solve(arr, n);
 for(int i=0; i<n; i++)
 {
   cout<<arr[i];
 }
 return 0;
}
```

```
solve(int arr[], int size)
 {
  arr[0]=100;
 }
```

Output

100 2 4

↳ **Remember that** :- While passing array as argument, we have to pass its size as an argument too because we don't have any explicit way to find its size.

↳ Here, size means no. of elements present in that array.

Q1. Find unique element in an array. Each element occurs twice except one.

Truth Table of XOR →

a		b		output
0	↔	0	→	0
0	↔	1	→	1
1	↔	0	→	1
1	↔	1	→	0

∴ OBSERVE - $0 \wedge a \rightarrow a$

XOR with '0' gives us the other element

```
int findUnique(int arr[], int size){
    int ans = 0;
    for(int i=0; i<size; i++){
        ans = ans ^ arr[i];
    }
    return ans;
}
int main(){
    int arr[] = {1,4,4,1,2,5,6,5,6};
    int size = 9;
    int finalAns = findUnique(arr, size);
    cout << "final ans = " << finalAns;
    return 0;
}
```

Output :-

final ans = 2

Explanation :-

Here, we are storing ans ^ arr[i] in ans variable i.e. after completing the loop ans will take XOR of every array element and calculate its XOR. Same values will be cancelled out and when the remaining value is going to calculate XOR with '0' then we get that unique element.

ans = 0 ^ 1 ^ 4 ^ 4 ^ 1 ^ 2 ^ 5 ^ 6 ^ 5 ^ 6

ans = 0 ^ 2 (same value will get cancelled or we can say, it will give '0' bcz XOR with same value will gives 0 in return)

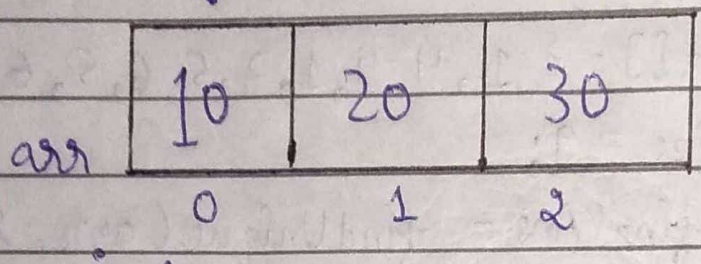
ans = 2

Page No.
Date

Q2. Print all pairs of an array.

Input Array ⟶ {10, 20, 30}

Output ⟶ (10,10), (20,10), (30,10)
(10,20), (20,20), (30,20)
(10,30), (20,30), (30,30)



for i=0, j=0 → print arr[i], arr[j] → (10,10)
j=1 → (10,20)
j=2 → (10,30)

i=1, j=0 → (20,10)
j=1 → (20,20)
j=2 → (20,30)

i=2, j=0 → (30,10)
j=1 → (30,20)
j=2 → (30,30)

∵ for every value of "i" we have to travers the complete array every time.

Code:-

```
void printPairs(int arr[], int size){
    for (int i=0; i<size; i++){
        for (int j=0; j<size; j++){
            cout << arr[i] << ", " << arr[j] << endl;
        }
        cout << endl;
    }
}
int main(){
    int arr[] = {10, 20, 30};
    int size = 3;
    //function call
    printPairs(arr, size);
```

Output:-

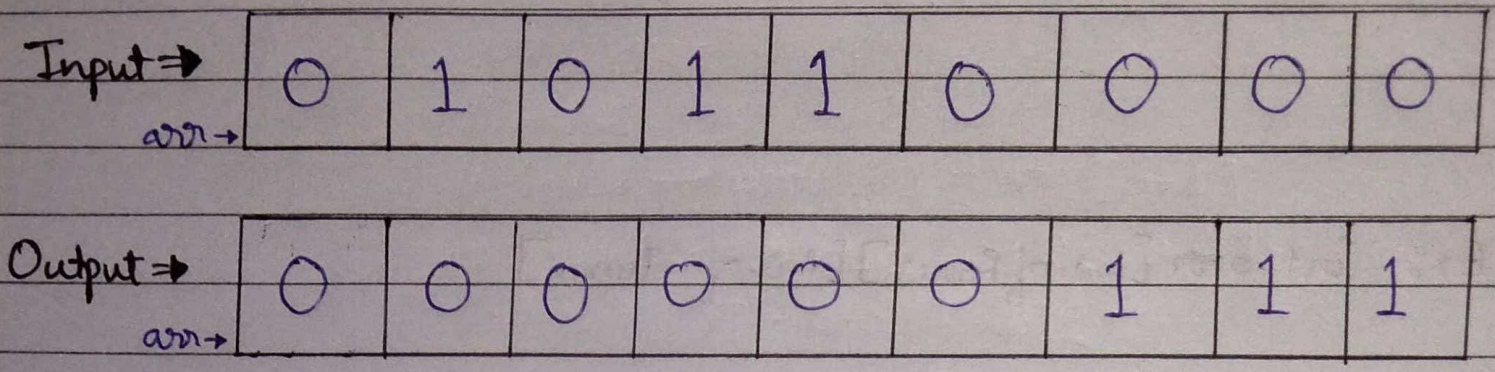
```
10, 10
10, 20
10, 30

20, 10
20, 20
20, 30

30, 10
30, 20
30, 30
```

Date ..../..../...

Q30. Sort 0's and 1's.



Approaches → ① Counting
→ ② 2 pointer approach → H/W
→ ③ sort() → covered in further lectures

① Counting :- Logic :- (i) Count 0 and 1
(ii) place 0
(iii) place 1

Code :-

```
void sortZeroOne(int arr[], int size){
    int zeroCount = 0;
    int oneCount = 0;
// Step A: Count zeros and ones
    for(int i=0; i<size; i++){
        if(arr[i] == 1){
            oneCount++;
        }
        if(arr[i] == 0){
            zeroCount++;
        }
    }
// place 0 and ones in array
    int i;
```

Date ..../..../.....

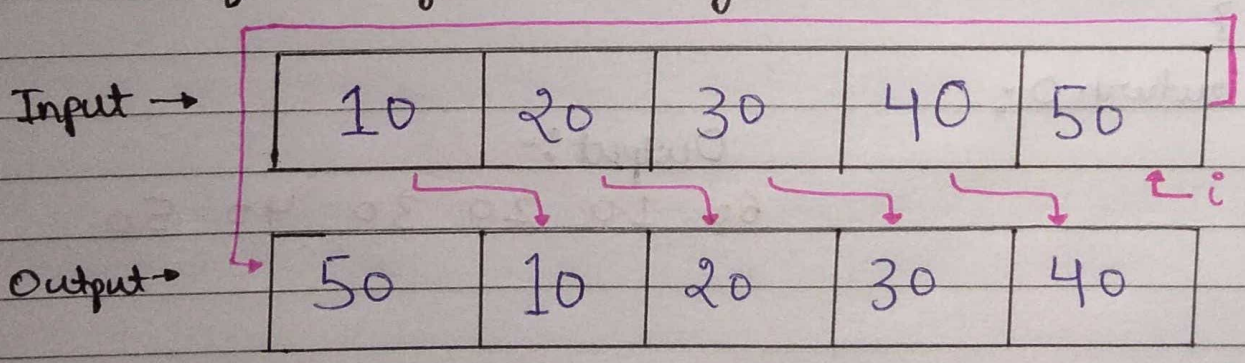
```
for(i=0; i<zeroCount; i++){
    arr[i]=0;
}

for(int j=i; j<size; j++){
    arr[j]=1;
}
}
int main(){
    int array[]={0,1,1,0,0,1,1,0,0};
    int n=9;
    // function call
    sortZeroOne(array, n);
    // Printing the array
    for(int i=0; i<n; i++){
        cout<<array[i]<<" ";
    }
    return 0;
}
```

Output :-

0 0 0 0 0 1 1 1 1

Q4. Shift arrays element by 1.



Logic :-

(A) Store last array value in a temp variable.
(B) Iterate from i=n-1 to i=1 & store value of 'i-1' in 'i'.
(C) Store value of temp in arr[0] i.e first place.

Spiral

Code:-

```
void shiftArray(int arr[], int n){
    // step 1
    int temp = arr[n-1];
    //step2
    // shift → arr[i] = arr[i-1]
    for(int i = n-1; i>=1; i--){
        arr[i] = arr[i-1];
    }
    // Step3 → copy temp into 0th index
    arr[0] = temp;
}

int main(){
    int arr[] = {10, 20, 30, 40, 50, 60};
    int size = 6;
    // function call
    shiftArray(arr, size);

    // printing the array
    for(int i=0; i<size; i++){
        cout << arr[i] << " ";
    }
    return 0;
}
```

Output:-

60 10 20 30 40 50