



Parameter passing techniques



Objectives

To learn and appreciate the following concepts:

- Parameter passing techniques
- Pass by value
- Pass by reference



Session outcome

At the end of session one will be able to understand:

- The overall ideology of parameter passing techniques

Review of pointers ...

Variable	Value	Address
Quantity	50	5000
p	5000	5048

int Quantity=50; //defines variable Quantity of type int

int* p; //defines p as a pointer to int

p = &Quantity; //assigns address of variable Quantity to pointer p

Now...

***p = 3;** //assigns 3 to Quantity

Functions-Overview:

```
void dispNum( int n ) // function definition
{
    printf(" The entered num=%d", n);
}

int main(){ //calling program
    int no;
    printf("Enter a number \n");
    scanf("%d",&no);
    dispNum( no); //Function reference
    return 0;
}
```

Formal parameters

Actual parameters



Functions- Parameter Passing

- Pass by value (call by value)
- Pass by reference (call by reference)

Pass by value:

```
void swap(int x, int y )
{
    int t=x;
    x=y;
    y=t;
    printf("In fn: x= %d and y=%d ",x,y);
}

int main()
{
    int a=5,b=7;
    swap(a, b);
    printf("After swap: a= %d and b= %d",a,b);
    return 0;
}
```

Output:

In fn: x = 7 & y = 5

After swap: a = 5 & b = 7

Pass by Reference – Using Pointers:

```
void swap(int *x, int *y )  
{  
    int t=*x;  
    *x=*y;  
    *y=t;  
}
```

Change is directly on the variable using the reference to the address.

When function is called:
address of a → x
address of b → y

Output:
After swap: a = 7 and b = 5

```
int main()  
{  
    int a=5,b=7;  
    swap(&a, &b);  
    printf("After swap: a=%d and b= %d",a,b);  
    return 0; }
```


Pointers as function arguments:

When we pass addresses to a function, the parameters receiving the addresses should be pointers.

```
int change (int *p)
{
    *p = *p + 10 ;
    return 0;
}
```

```
#include <stdio.h>

int main() {
    int x = 20;

    change(&x);

    printf("x after change==%d",x);

    return 0;
}
```

Output :
x after change=30

Pointers as function arguments

- When the function `change()` is called, the address of the variable `x`, not its value, is passed into the function `change()`.
- Inside `change()`, the variable `p` is declared as a pointer and therefore `p` is the address of the variable `x`. The statement
- `*p=*p +10` adds 10 to the value stored at the address `p`. Since `p` represents the address of `x`, the value of `x` is changed from 20 to 30. therefore it prints 30.

Function that return multiple values-Using pointers

Using pass by reference we can write a function that return multiple values.

```
void fnOpr(int a, int b, int *sum, int *diff) {  
    *sum = a + b;  
    *diff = a - b; }  

```

```
int main() {  
    int x, y, s, d;  
    printf("Enter two numbers: \n");  
    scanf("%d %d",&x, &y);  
    fnOpr(x, y, &s, &d);  
    printf("Sum = %d & Diff =%d ", s, d);  
    return 0;  
}
```

Output:

x= 5 & y= 3
Sum =8 & Diff = 2

Nesting of functions:

- C language allows nesting of functions by calling one function inside another function.
- Nesting of function does not mean that we can define an entire function inside another function. The following examples shows both valid and invalid function nesting in C language

// Wrong way of function nesting

```
void fun()
{
    printf("I am having Fun....");

    void sleep()
    {
        printf("I am having sleep");
    }
}
```

// Right way of function nesting

```
void sleep() {
    printf("I am having sleep");
}

void fun()
{
    printf("I am having Fun....");
    sleep();
}
```

Nesting of Functions:

```
void First (void){ // FUNCTION DEFINITION
    printf("I am now inside function First\n");
}

void Second (void){ // FUNCTION DEFINITION
    printf( "I am now inside function Second\n");
    First(); // FUNCTION CALL
    printf("Back to Second\n");
}

int main (){
    printf( "I am starting in function main\n");
    First (); // FUNCTION CALL
    printf( "Back to main function \n");
    Second (); // FUNCTION CALL
    printf( "Back to main function \n");
    return 0;
}
```

Nesting of Functions:

```
void fnOpr(int a, int b, int *sum, int *diff)
{
    *sum = a + b;
    if (fnDiff(a,b))
        *diff = a -b;
    else
        *diff = b - a;
}

int main() {
    int x, y, s, d;
    printf("Enter the values: \n");
    scanf("%d %d", &x, &y);
    fnOpr(x, y, &s, &d);
    printf("The results are, Sum =%d and Diff = %d", s, d);
    return 0; }
```

```
int fnDiff(int p, int q){
    if (p>q)
        return(1);
    else
        return (0);
}
```

Output:

```
x= 3 & y= 5
s =8 & d = 2
```

Passing 1D-Array to Function:


```
int fnSum( int a[ ], int n) {  
    int sum=0,i;  
    for(i=0;i<n;i++)  
        sum+=a[i];  
    return (sum); }  

```

Output: n=5
1, 2, 3, 4, 5
Sum of elements = 15

```
int main() {  
    int n, a[20], x, y,i;  
    printf("Enter the limit \n");  
    scanf("%d",&n);  
    printf("Enter the values: \n");  
    for (i=0; i<n; i++)  
        scanf("%d",&a[i]);  
    printf("The sum of array elements is =%d ",fnSum(a, n));  
    return 0;  
}
```

Array name is passed along
with number of elements



Passing 1D-Array to Function

Rules to pass an array to a function

- The function must be called by passing only the **name of the array**.
- In the function definition, the formal parameter **must be** an array type; the **size of the array** does not need to be specified.
- The function prototype must show that argument is an array.

Passing 2D-Array to Function:

```
int fn2d(int x[ ][10], int m, int n)
{
    int i, j, sum=0;
    for(i=0; i<m; i++)
        for(j=0; j<n; j++)
            sum+=x[i][j];
    return (sum) ;
}
```

Output: m=2 n=3

1 2

3 4

5 6

Sum of elements = 21

```
int main() {
    int i, j, a[10][10], m, n;
    printf("Enter dimentions of matrix");
    scanf("%d%d", &m, &n);
    printf("Enter the elements");
    for(i=0;i<m;i++)
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    printf ("Sum of elements is=%d",fn2d(a, m, n));
    return 0;
}
```

Passing 2D-Array to Function:

Rules to pass a 2D- array to a function

- The function must be called by passing only the **array name**.
- In the function definition, we must indicate that the array has two-dimensions by including **two set of brackets**.
- The size of the second dimension **must be specified**.
- The prototype declaration should be similar to function header.

Problems:

- Write a c program to add all the even elements of an array using a function Add().
- Write a C program to replace all odd numbers of an array with the largest number in the array using a function Replace().
- Write a C program to replace all the zeros in the matrix by the trace of the matrix using a function Trace().
- Write a C program using pass-by-pointer method to compute the compound interest using a function Compound().

Write a c program to add all the even elements of an array using a function Add()

```
int AddE( int a[ ], int n)
{
    int sum=0,i;
    for(i=0;i<n;i++)
    {
        if((a[i]%2) == 0)
            sum+=a[i];
    }
    return (sum);
}
```

```
int main()
{
    int n, a[20], x, y, i;
    printf("Enter the limit \n");
    scanf("%d", &n);
    printf("Enter the values: \n");
    for (i=0; i<n; i++)
        scanf("%d", &a[i]);
    printf("The sum of even array
           elements is =%d ",AddE(a,n));

    return 0;
}
```

Write a C program to replace all odd numbers of an array with the largest number in the array using a function Replace()

```
void Replace( int arr[ ], int n)
{
    // To find the largest
    int i, max = arr[0];
    for (i = 1; i < n; i++)
        if (arr[i] > max)
            max = arr[i];

    // To replace
    for (i=0; i<n; i++) {
        if (arr[i]%2 != 0)
            arr[i]=max;
    }
}
```

```
int main() {
    int n, a[20], x, y,i;
    printf("Enter the limit \n");
    scanf("%d",&n);

    printf("Enter the values: \n");
    for (i=0; i<n; i++)
        scanf("%d",&a[i]);

    Replace(a, n);
    printf("The array after replacement is\n");
    for (i=0; i<n; i++)
        printf("%d \n",a[i]);

    return 0;
}
```

Write a C program to replace all the zeros in the matrix with norm of the matrix using a function Norm()

```
#include <math.h>
void Norm1(float a[ ][10], int m, int n) {
    int i, j, norm, sum=0;

    // Finding Trace
    for(i=0;i<m;i++) {
        for(j=0;j<n;j++)
            sum=sum+a[i][j]*a[i][j];
    }
    norm=sqrt(sum);
    //Replacing zeros
    for(i=0;i<m;i++)
        for(j=0;j<n;j++)
            if(a[i][j]==0)
                a[i][j]=norm;
}
```

```
int main() {
    int i, j, m, n;
    float a[10][10];
    printf("Enter dimention of matrix");
    scanf("%d %d", &m, &n);
    printf("Enter the elements");
    for(i=0;i<m;i++)
        for(j=0;j<n;j++)
            scanf("%f",&a[i][j]);
    Norm1(a, m, n);
    printf("Matrix after replacement \n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%f",a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Summary:

- Parameter passing techniques

- pass by value

- void swap(int x, int y)**

- pass by reference

- void swap(int *x, int *y)**

- Passing 1 D array

- int fnParr(int a[], int n)**

- Passing 2 D array

- int fn2d(int x[][10], int m, int n)**