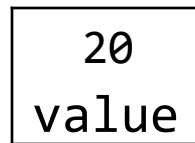


Pass by value
&
Pointer

Preliminaries

a



1004
address

Actual Parameters

swap(10, 20)

Function Call

Formal Parameters

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}
```

Function Définition

Pass by Value

- Actual parameter and formal parameter refer to **different** memory locations
- Any changes made to the formal parameter **will not get reflected** in actual parameter
This is true whether the variables have the same name in both functions or whether the names are different
- the numerical value is passed to the function, and used to initialize the values of the functions formal parameters.
- Ordinary data types ints, floats, doubles, chars etc. are **passed by value**

Function Call

```
int a= 10, b = 20;  
swap(a, b)
```

Function Définition

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}
```

Pass by Value

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
int main() {  
    int a = 10, b = 20;  
    swap(a, b); //call by value  
}
```

1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Value

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
int main() {  
    int a = 10, b = 20;  
    swap(a, b); //call by value  
}
```

a					b			
	10					20		
1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Value

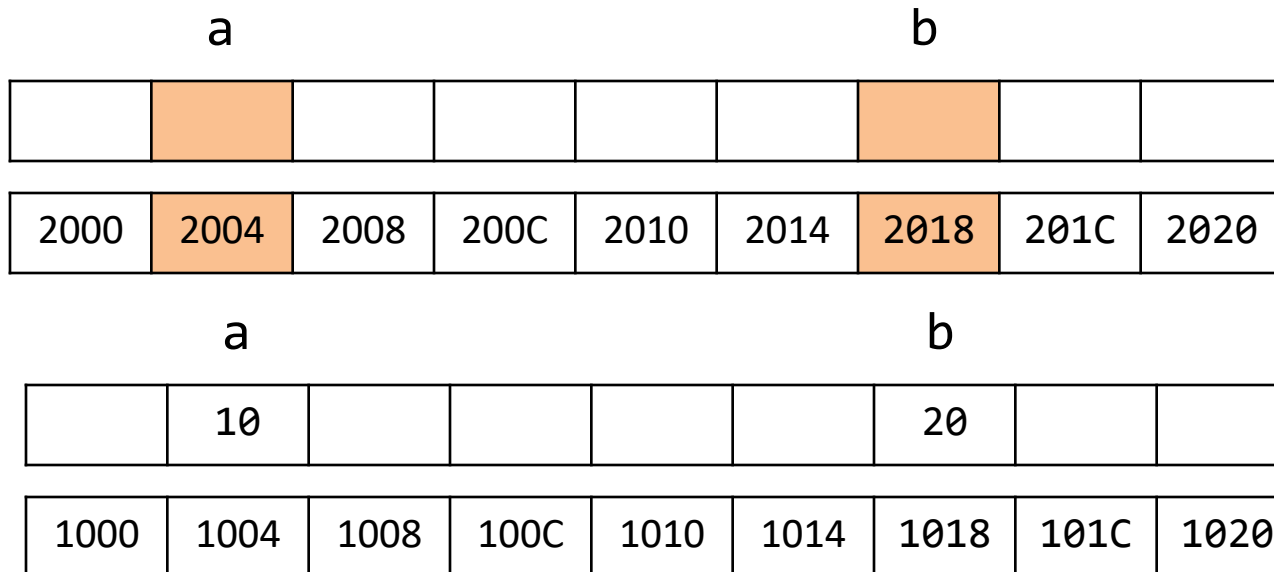
```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
int main() {  
    int a = 10, b = 20;  
    swap(a, b); //call by value  
}
```

a					b			
	10					20		
1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Value

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
int main() {  
    int a = 10, b = 20;  
    swap(a, b); //call by value  
}
```

Actual parameter and formal parameter refer to **different** memory locations even if they have the same name



Pass by Value

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
int main() {  
    int a = 10, b = 20;  
    swap(a, b); //call by value  
}
```

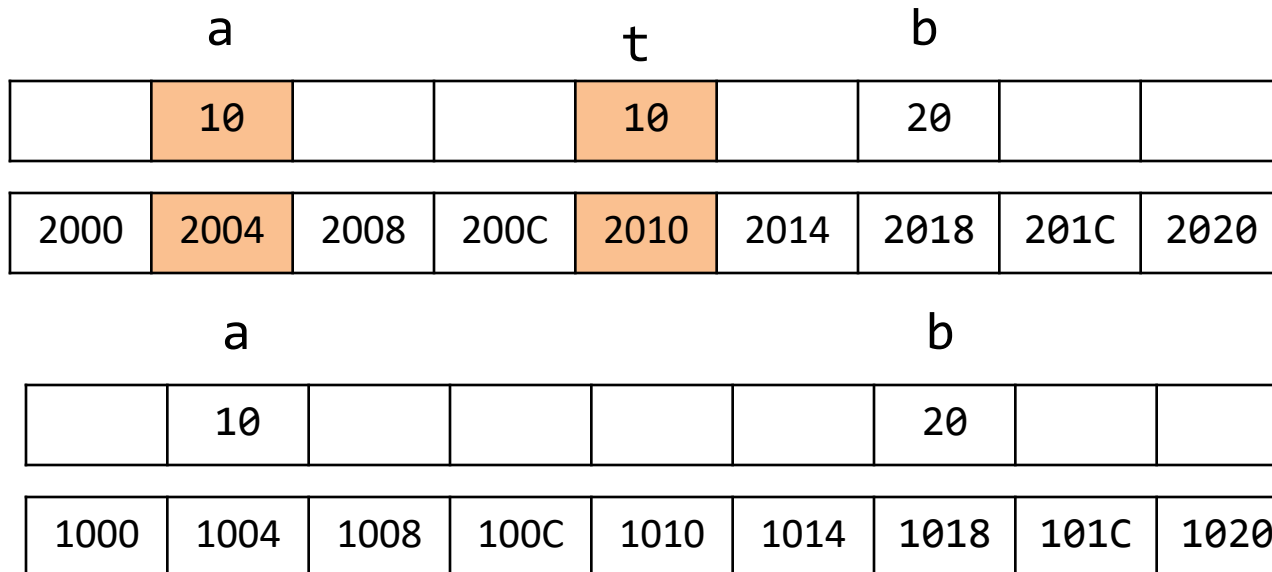
numerical value is passed to the function, and used to initialize the values of the functions formal parameters

a					b			
	10					20		
2000	2004	2008	200C	2010	2014	2018	201C	2020

a					b			
	10					20		
1000	1004	1008	100C	1010	1014	1018	101C	1020

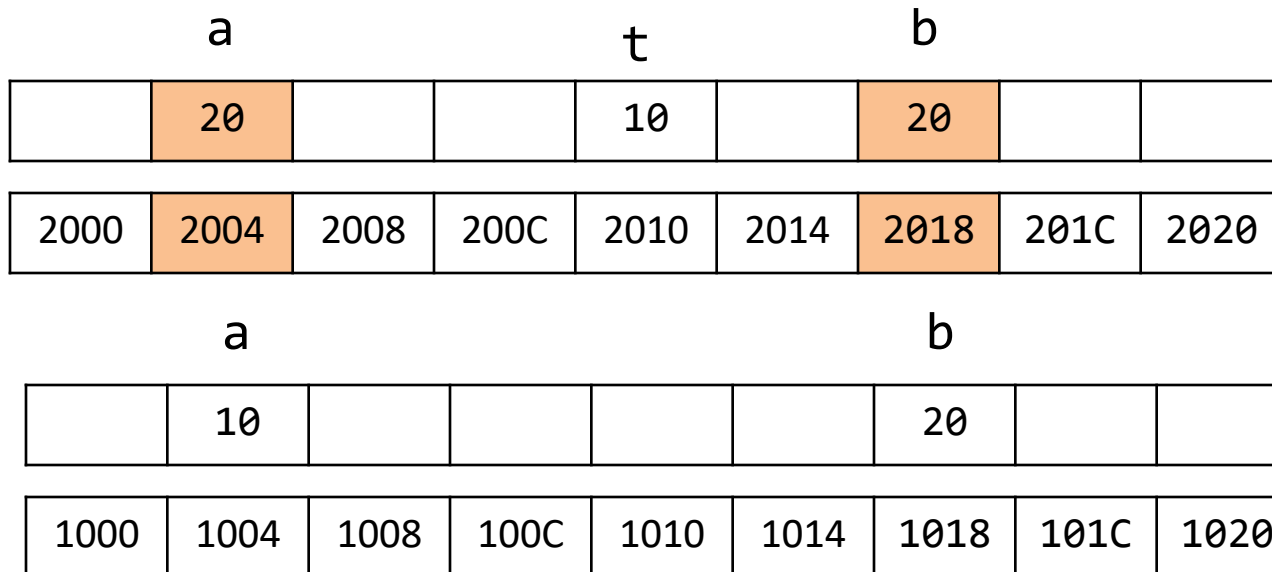
Pass by Value

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
int main() {  
    int a = 10, b = 20;  
    swap(a, b); //call by value  
}
```



Pass by Value

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
int main() {  
    int a = 10, b = 20;  
    swap(a, b); //call by value  
}
```



Pass by Value

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
int main() {  
    int a = 10, b = 20;  
    swap(a, b); //call by value  
}
```

a				t	b			
	20			10		10		
2000	2004	2008	200C	2010	2014	2018	201C	2020

a					b			
	10					20		
1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Value

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
int main() {  
    int a = 10, b = 20;  
    swap(a, b); //call by value  
}
```

a					b			
	10					20		
1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Value

```
void swap(int a, int b) {  
    int t = a;  
    a = b;  
    b = t;  
}  
int main() {  
    int a = 10, b = 20;  
    swap(a, b); //call by value  
    printf("%d, %d", a, b);  
}
```

```
>> Console  
10 20
```

a					b			
	10					20		
1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Pointer/Address

- Actual parameter and formal parameter refer to **same** memory locations
- Any changes made to the formal parameter **will get reflected** in actual parameter
- Addresses are passed to the functions

Function Call

```
int a= 10, b = 20;  
swap(&a, &b)
```

Function Définition

```
void swap(int *a, int *b) {  
    int t = *a;  
    *a = *b;  
    *b = t;  
}
```

Pass by Pointer/Address

```
void swap(int *a, int *b) {  
    int t = *a;  
    *a = *b;  
    *b = t;  
}  
int main() {  
    int a= 10, b = 20;  
    swap(&a, &b); //call by ptr  
}
```

1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Pointer/Address

```
void swap(int *a, int *b) {  
    int t = *a;  
    *a = *b;  
    *b = t;  
}  
int main() {  
    int a= 10, b = 20;  
    swap(&a, &b); //call by ptr  
}
```

a				b				
	10						20	
1000	1004	1008	100C	1010	1014	1018	101C	1020

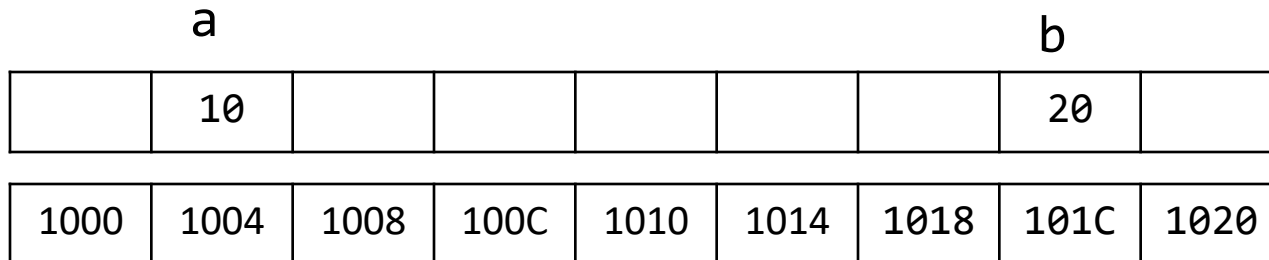
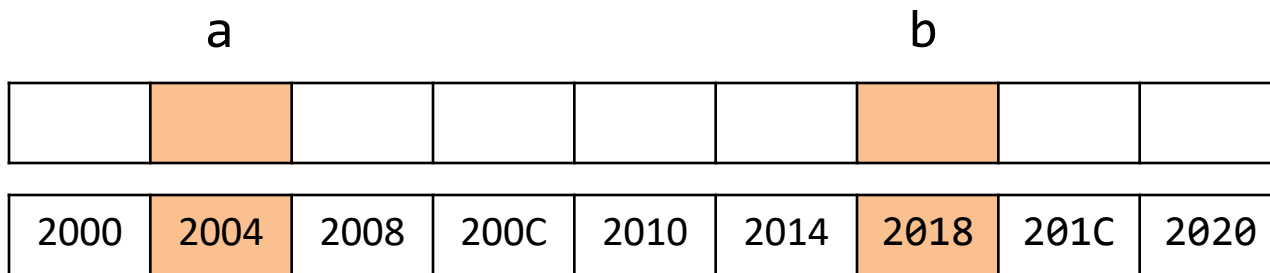
Pass by Pointer/Address

```
void swap(int *a, int *b) {  
    int t = *a;  
    *a = *b;  
    *b = t;  
}  
int main() {  
    int a= 10, b = 20;  
    swap(&a, &b); //call by ptr  
}
```

a				b				
	10						20	
1000	1004	1008	100C	1010	1014	1018	101C	1020

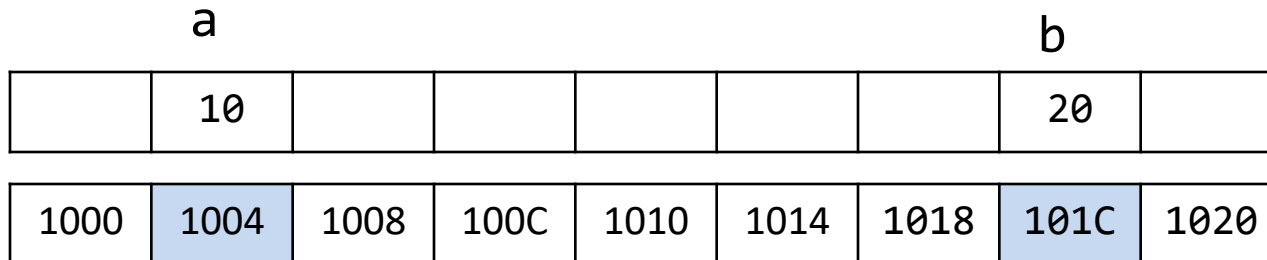
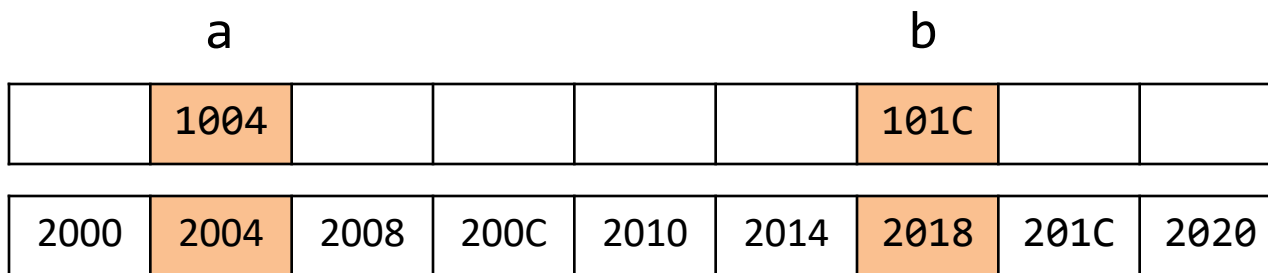
Pass by Pointer/Address

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;  
    *b = t;  
}  
int main() {  
    int a= 10, b = 20;  
    swap(&a, &b); //call by ptr  
}
```



Pass by Pointer/Address

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;  
    *b = t;  
}  
int main() {  
    int a= 10, b = 20;  
    swap(&a, &b); //call by ptr  
}
```



Pass by Pointer/Address

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;  
    *b = t;  
}  
int main() {  
    int a= 10, b = 20;  
    swap(&a, &b); //call by ptr  
}
```

a					b			t
	1004				101C			10
2000	2004	2008	200C	2010	2014	2018	201C	2020

a					b			
	10					20		
1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Pointer/Address

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;  
    *b = t;  
}  
int main() {  
    int a= 10, b = 20;  
    swap(&a, &b); //call by ptr  
}
```

a					b			t
	1004				101C			10
2000	2004	2008	200C	2010	2014	2018	201C	2020

a						b		
	20						20	
1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Pointer/Address

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;  
    *b = t;  
}  
int main() {  
    int a= 10, b = 20;  
    swap(&a, &b); //call by ptr  
}
```

a					b		t	
	1004					101C		10
2000	2004	2008	200C	2010	2014	2018	201C	2020

a						b		
	20						10	
1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Pointer/Address

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;  
    *b = t;  
}  
int main() {  
    int a= 10, b = 20;  
    swap(&a, &b); //call by ptr  
}
```

a							b	
	20						10	
1000	1004	1008	100C	1010	1014	1018	101C	1020

Pass by Pointer/Address

```
void swap(int *a, int *b){  
    int t = *a;  
    *a = *b;  
    *b = t;  
}  
int main() {  
    int a= 10, b = 20;  
    swap(&a, &b); //call by ptr  
    printf("%d, %d", a, b);  
}
```

```
>> Console  
20 10
```

a				b				
	20						10	
1000	1004	1008	100C	1010	1014	1018	101C	1020

Passing Arrays

- When the entire arrays are passed, it is done by pass by pointer/address
- Any changes made to the array in user-defined function will be reflected in main function or the function from which it is called

Function Call

```
int a[5] = {1, 2, 3, 4, 5};  
f(a)
```

Function Définition

```
void f(int a[], int size) {  
  
}
```

Passing Arrays

```
void shift(int a[], int size, int num){  
    int i;  
    for(i = 0; i < size; i++){  
        a[i] = a[i] + num;  
    }  
}  
  
int main() {  
    int x[5] = {1, 2, 3, 4, 5};  
    shift(x, 5, 10);  
}
```



Address of the first element of array x

Passing Arrays

```
void shift(int a[], int size, int num){  
    int i;  
    for(i = 0; i < size; i++){  
        a[i] = a[i] + num;  
    }  
}  
  
int main() {  
    int x[5] = {1, 2, 3, 4, 5};  
    shift(x, 5, 10);  
}
```

1000	1004	1008	100C	1010	1014	1018	101C	1020

Passing Arrays

```
void shift(int a[], int size, int num){  
    int i;  
    for(i = 0; i < size; i++){  
        a[i] = a[i] + num;  
    }  
}  
  
int main() {  
    int x[5] = {1, 2, 3, 4, 5};  
    shift(x, 5, 10);  
}
```

x

			1	2	3	4	5	
1000	1004	1008	100C	1010	1014	1018	101C	1020

Passing Arrays

```
void shift(int a[], int size, int num){  
    int i;  
    for(i = 0; i < size; i++){  
        a[i] = a[i] + num;  
    }  
}  
  
int main() {  
    int x[5] = {1, 2, 3, 4, 5};  
    shift(x, 5, 10);  
}
```

x

			1	2	3	4	5	
1000	1004	1008	100C	1010	1014	1018	101C	1020

Passing Arrays


```
void shift(int a[], int size, int num){  
    int i;  
    for(i = 0; i < size; i++){  
        a[i] = a[i] + num;  
    }  
}
```

```
int main() {  
    int x[5] = {1, 2, 3, 4, 5};  
    shift(x, 5, 10);  
}
```

x

			11	12	13	14	15	
1000	1004	1008	100C	1010	1014	1018	101C	1020

Passing Arrays

```
void shift(int a[], int size, int num){  
    int i;  
    for(i = 0; i < size; i++){  
        a[i] = a[i] + num;  
    }  
}  
  
int main() {  
    int x[5] = {1, 2, 3, 4, 5};  
    shift(x, 5, 10);  
      
}
```

x

			11	12	13	14	15	
1000	1004	1008	100C	1010	1014	1018	101C	1020

const keyword

- prevents a function from changing the array values

```
void shift(const int a[], int size, int num){  
    int i;  
    for(i = 0; i < size; i++){  
        a[i] = a[i] + num;  
    }  
}
```

```
int main() {  
    int x[5] = {1, 2  
    shift(x, 5, 10);  
}
```

Generates an error message
as array a is declared as constant
You can not change the elements of it

Using Arrays to Return Multiple Values

- You can use arrays to return multiple values from a user-defined function as they are passed by pointers

```
void calculator(int a, int b, int c[]){
    c[0] = a + b;
    c[1] = a - b;
    c[2] = a * b;
    c[3] = a / b;
}

int main() {
    int a = 10, b = 20;
    int cal[4];
    calculator(a, b, cal);
    printf("Sum = %d\n", cal[0]);
    printf("Sub = %d\n", cal[1]);
    printf("Mul = %d\n", cal[2]);
    printf("Div = %d\n", cal[3]);
    return 0;
}
```

We pass array `cal` to function `calculator` so that we can save the sum, subtraction, multiplication and division in the array which can be accessed from main after the function call