

Structure as Parameter

call by value

```
struct Rectangle {
```

```
    int length;
```

```
    int breadth;
```

```
};
```

```
int area (struct Rectangle r1) {
```

```
    return r1.length * r1.breadth;
```

```
}
```

```
int main() {
```

```
    struct Rectangle r = {10, 5};
```

```
    printf("%d", area(r));
```

```
}
```

r_1
length 10
breadth 5

r/r_1
length 10
breadth 5

```
→ int area (struct Rectangle &r1) {
```

```
    return r1.length * r1.breadth;
```

```
}
```

Call by reference

Call by Address

```
struct Rectangle {
```

```
    int length;
```

```
    int breadth;
```

```
};
```

```
void changeLength(struct Rectangle *p, int L) {
```

```
    *p->length = L;
```

```
}
```

```
int main() {
```

```
    struct Rectangle r = {10, 5};
```

```
    changeLength(&r, 20);
```

```
    cout << r.length << endl;
```

```
}
```

Diagram illustrating the call by address mechanism:

A pointer variable `p` (represented by a box with `p` above it) is shown. An arrow points from `p` to a memory location containing a `Rectangle` structure. This structure is represented as a table:

length	20
breadth	5

The value 20 in the `length` field is crossed out and replaced with 20, indicating the modification made by the function.

Array inside a structure;

```
struct Test {
```

```
    int A[5];
```

```
    int n;
```

```
}
```

```
void func(struct Test t1) {
```

```
    t1.A[0] = 10;
```

```
    t1.A[1] = 9;
```

```
}
```

```
int main() {
```

```
    struct Test t = { {2, 4, 6, 8, 10}, 5};
```

```
    fun(t);
```

```
}
```

array is called by value

t1				
10 2	9 4	6	8	10
5				

t

A	2	4	6	8	10
n	5				

array won't be modified

but can't be

modified as it called by value