

Exercise 6-2: callby.c

Maximilian Fernaldy - C2TB1702

Note: some links and other HTML-related objects may not work in pdf form. Consider reading the webpage format of the report [here](#).

Exercise 6-2.Difference in Calls

15

■ Write and execute program callby.c

```
#include <stdio.h>
/* prototype declaration */
int add(int v);
void addp(int *p);

int main() {
    int a,b;

    printf("Input value: ");
    scanf("%d", &a);

    b = add(a);
    printf("Call by value: %d\n", b);
    b = a;
    addp(&b);
    printf("Call by reference: %d\n", b);
    return 0;
}

/* call by value function */
int add(int v) {
    v += 10; // add 10
    return v;
}

/* call by reference(pointer) */
void addp(int *p) {
    *p += 10; // add 10
}
```

value
reference

main function

a b

&b

address

add function

b result

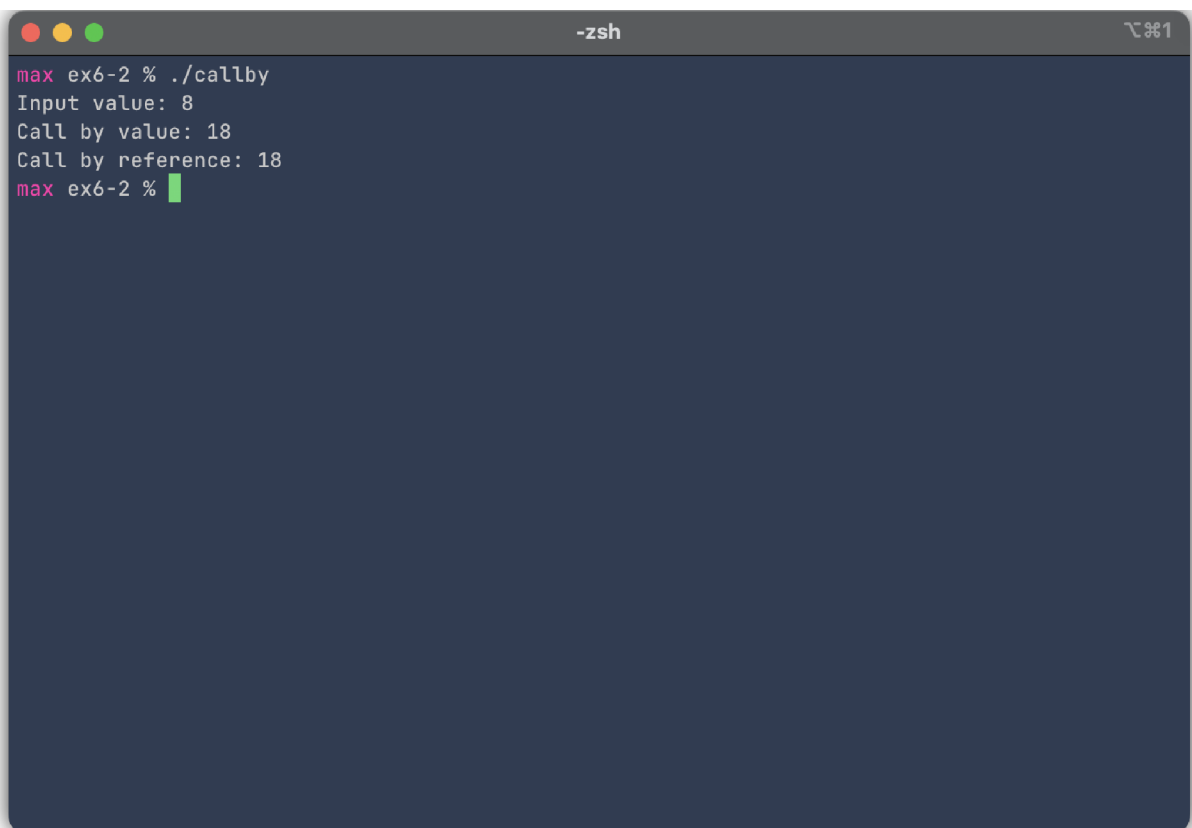
Since there is an int type variable b at the end of the sent pointer, the result can be assigned to the contents (* p) of the pointer.

callby.c highlights two methods to modify a variable declared in the main function. The first method `add()` copies the value of the variable, passes it into the function for modification, then assigns the value that is returned by the function to the variable. The second method `addp()` passes the memory address of the variable, or in other words a pointer to the function, and the function modifies the variable directly by accessing it through its memory address.

Notice how when `add()` is called, the variable `a` is passed into the function, which passes its value as the argument. However, when `addp()` is called, the address `&b` is passed instead, and in the function the address is stored in the pointer `p`, which is then used to access the variable directly by dereferencing it with `*p` then adding 10 to the variable's value.

`add()` works with a copy of `a`, while `addp()` works directly with the variable `b`. This is the main difference between the two. When we pass a copy of the variable's value with `add(a)`, we are allocating additional memory for the copy, because the intended use for this method is to modify it separately of the variable in the main function. As a result, modifications to the copy will not be reflected in the original, except if the modifications are applied by assigning the function's return value to the original variable. Conversely, When we pass the memory address of the variable, although it looks like we're doing the same thing in a different way, this method is fundamentally different. When a variable is created, the memory address will be reserved as well, whether or not we use it in the future. When we use that memory address to pass into a function, we are working directly with the variable, which means the function doesn't allocate any additional memory, and any changes made to the variable will be immediately reflected in the main function.

Running the program shows that the two methods return the same answer:

A terminal window with a dark blue background and a grey title bar. The title bar contains three colored window control buttons (red, yellow, green) on the left, the text '-zsh' in the center, and a window icon and number '1' on the right. The terminal displays the following text: 'max ex6-2 % ./callby' on the first line, 'Input value: 8' on the second line, 'Call by value: 18' on the third line, 'Call by reference: 18' on the fourth line, and 'max ex6-2 %' on the fifth line followed by a green cursor block.

```
max ex6-2 % ./callby
Input value: 8
Call by value: 18
Call by reference: 18
max ex6-2 %
```

The same result is returned for both functions