

Functions & Pointers (1A)

Copyright (c) 2010 – 2016 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Call by Value

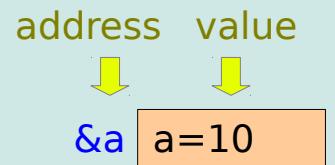
```
void func(int n);
```

```
int main (void)
{
    int a = 10;

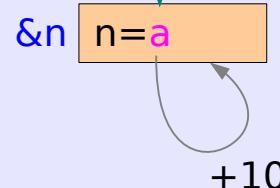
    printf("a = %d \n", a);
    func (a);
    printf("a = %d \n", a);

    return 0;
}
```

```
void func (int n)
{
    printf("n = %d \n", n);
    n += 10;
    printf("n = %d \n", n);
}
```



the **value** of **a** is passed through the parameter variable **n**



n is **local** to the function **func** and exists **while** the function is being **called**

Call by Reference

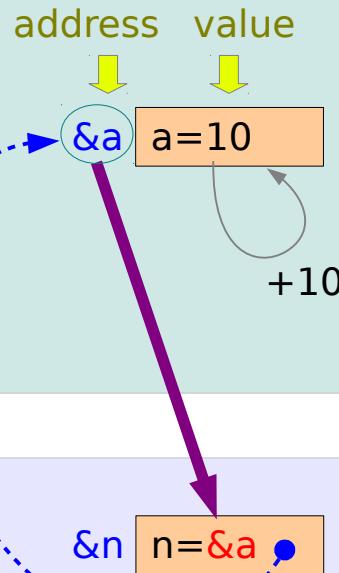
```
void func(int * n);
```

```
int main (void)
{
    int a = 10;

    printf("a = %d \n", a);
    func (&a);
    printf("a = %d \n", a);

    return 0;
}
```

```
void func(int * n)
{
    printf("*n = %d \n", *n);
    *n += 10;
    printf("*n = %d \n", *n);
}
```



the **address** of **a** is passed through the parameter variable **n**

***n += 10;**

n is **local** to the function **func** and exists **while** the function is being **called**

Call by Reference – Passing Arrays

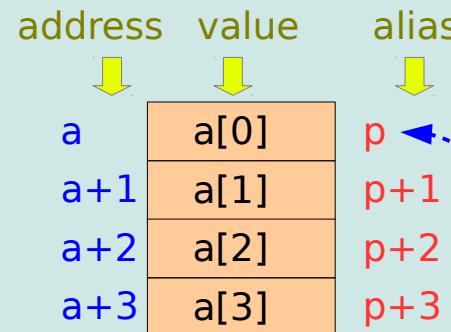
```
void sum(int p [], int n);
```

```
int main (void)
{
    int a[4], T;

    T = sum(a, 4);

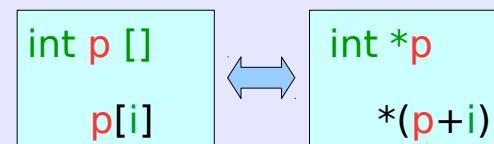
    printf("Sum = %d \n", T);

    return 0;
}
```



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

&p p=a ●



Function Calls in C (1)

int x, y;

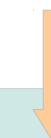
Data is passed

call by value

x



y



input

```
int func(int a, int b)
```

Address is passed

call by reference

&x

&y

in/out



```
int func(int * a, int * b )
```

Function Calls in C (2)

int *m, *n;

Data is passed

call by value

*m

*n

input



```
int func(int a, int b)
```

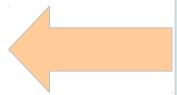
Address is passed

call by reference

m

n

in/out



```
int func(int * a, int * b )
```

Function Calls in C (3)

int x, y;

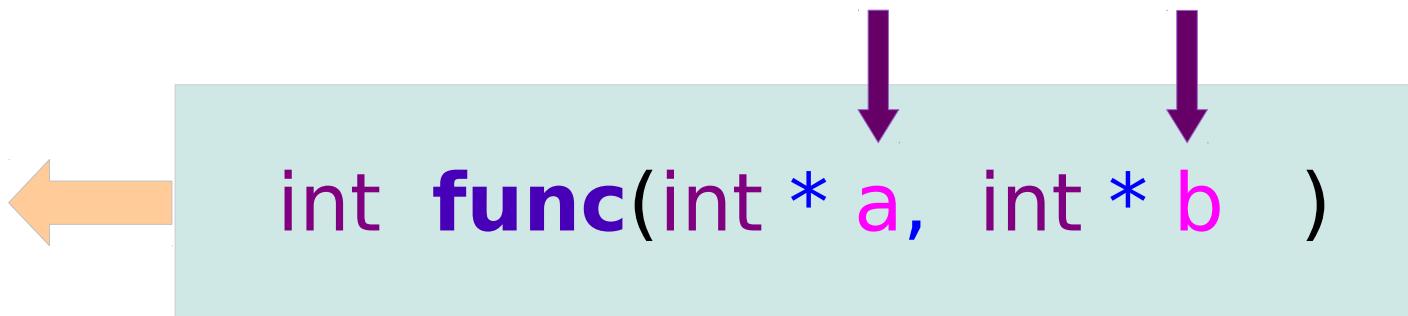
Address is passed

call by reference

$\&x$

$\&y$

in/out



input

RD $\leftarrow *a$ $\leftarrow x$

output

WR $\rightarrow *a$ $\rightarrow x$

input

RD $\leftarrow *b$ $\leftarrow y$

output

WR $\rightarrow *b$ $\rightarrow y$

Function Calls in C (4)

int *m, *n;

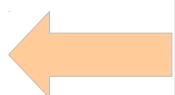
Address is passed

call by reference

m

n

in/out



int func(int * a, int * b)

input

RD

↔ *a

↔ *m

output

WR

→ *a

→ *m

input

RD

↔ *b

↔ *n

output

WR

→ *b

→ *n

A point to a function (1)

{ **int** **m** ; an integer variable
int *** m** ; a pointer variable

The diagram illustrates the relationship between a function prototype and a function pointer. It consists of two rows. The top row shows the declaration of a function named **f** that takes two integer parameters, **a** and **b**, and returns an integer. This is labeled as a "function's type". The bottom row shows the declaration of a function pointer named ***f** that also takes parameters **a** and **b** and returns an integer. This is labeled as a "function pointer". A green curly brace on the left side groups both declarations. A blue double-headed vertical arrow is positioned between the two rows, indicating they represent equivalent concepts.

int **f** (int a, int b); a prototype a function's type

int **(* f)** (int a, int b); a function pointer

without () → **int * f (int a, int b);**
a function's return type

A point to a function (2)

int **fn** (int a, int b); a function name fn



int **(* fp)** (int a, int b); a function pointer fp

explicit method

```
fp = &fn;
```

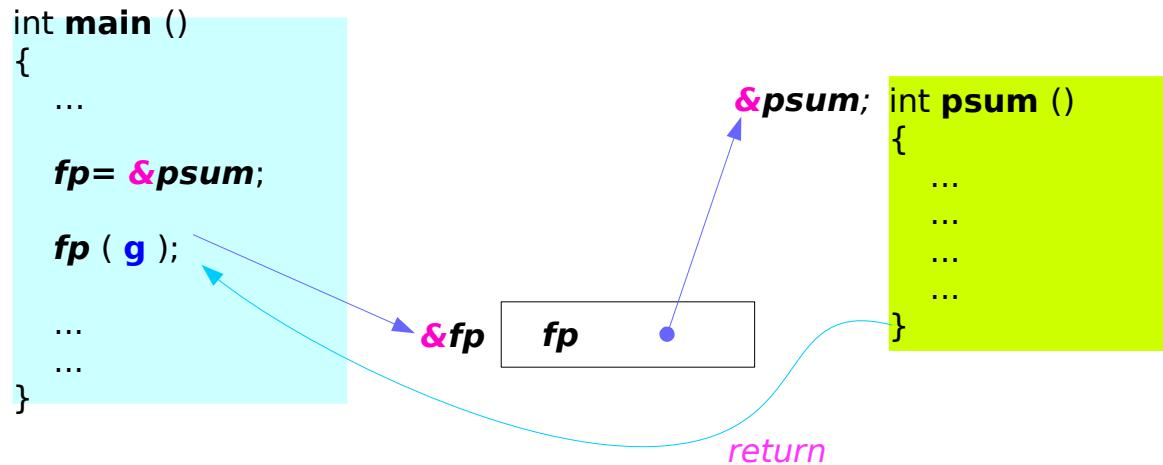
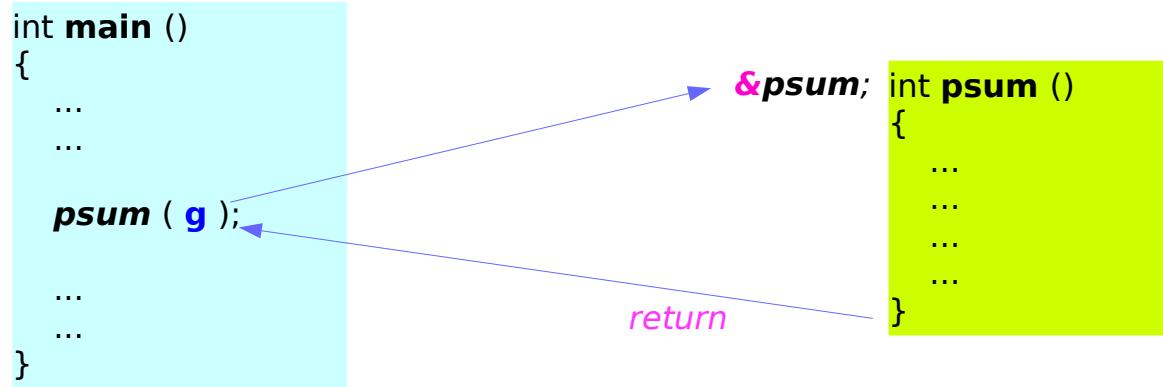
```
z = (*fp) (x, y);
```

implicit method

```
fp = fn;
```

```
z = fp (x, y);
```

Indirect Function Call



References

- [1] Essential C, Nick Parlante
- [2] Efficient C Programming, Mark A. Weiss
- [3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
- [4] C Language Express, I. K. Chun