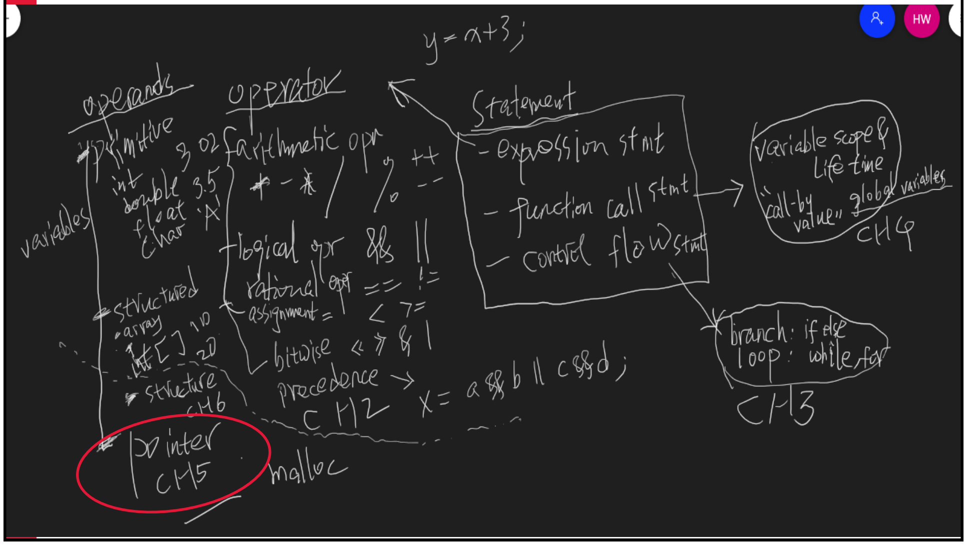
## Roadmap -- How the topics are related



3

# Pointers K&R Ch 5

- Basics: Declaration and assignment (5.1)
- Pointer to Pointer (5.6)
- Pointer and functions (5.2)
- Pointer arithmetic (5.4)
- Pointers and arrays (5.3)
- Arrays of pointers (5.6)
- Command line argument (5.10)
- Pointer to arrays and two dimensional arrays (5.9)
- Pointer to functions (5.11)
- Pointer to structures (6.4)
- Memory allocation (extra)

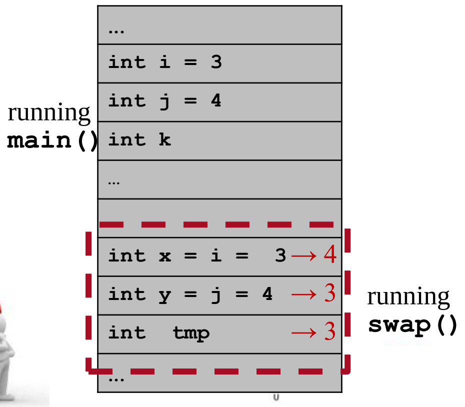
4

## Motivations: Pass-by-Value

RECALL

- In C, all functions are **pass by value**
  - Value of the arguments are passed to functions, but not the arguments themselves (i.e., not "~~pass by reference~~")

```
void swap (int x, int y)
{ int tmp;
  tmp = x;
  x = y;
  y = tmp;
}
main(){
  int i=3, j=4;
  swap(i,j)
}
```



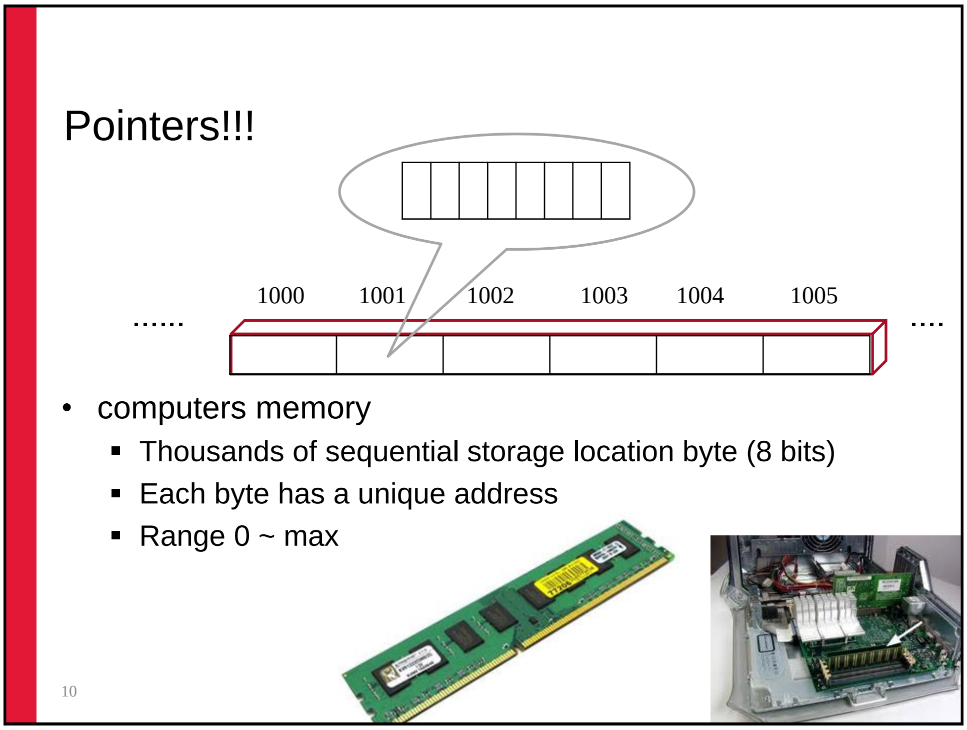
```
char fromStr [] = "Hello!";
char toStr [20];

strcpy(toStr, fromStr);     // toStr modified

fgets(toStr, 10, stdin);    // toStr modified
```

- Given an array as an argument, a function can modify the contents of the array -- Arrays are passed *as if* "call-by-reference"

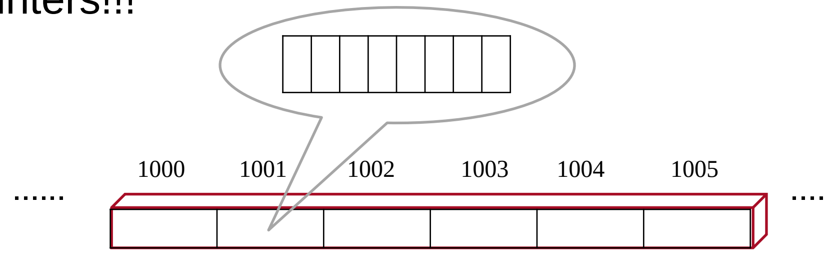
- But isn't C "call-by-value"? -- pass single numerical value
  - How to pass strings to **strcpy()**?
  - How does **strcpy(),scanf(),fgets()** modify argument?

- Also **scanf ("%d %s", &a, arr); // a arr modified**
  - Why **&a**, why not **&arr**

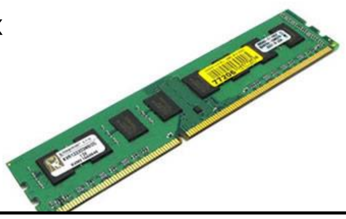
- Why **sizeof** does not work in function call
  - return 8 or 4 always



## Pointers!!!

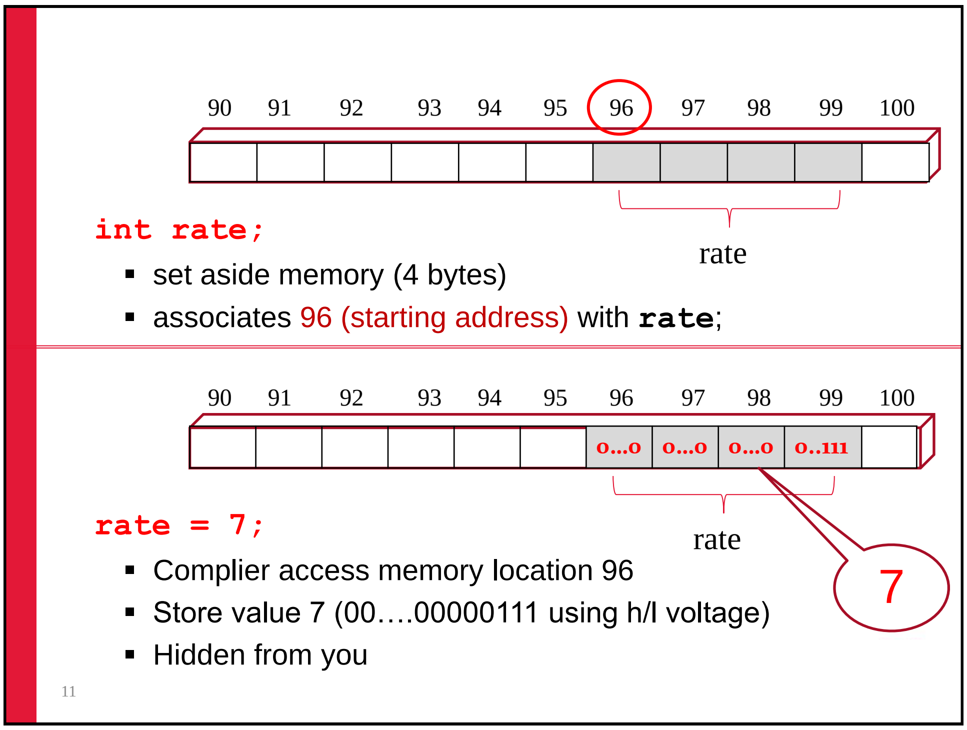


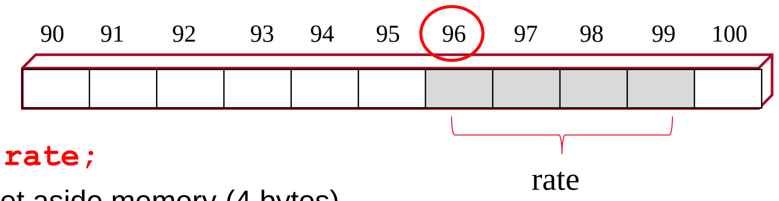
- computers memory
  - Thousands of sequential storage location byte (8 bits)
  - Each byte has a unique address
  - Range 0 ~ max



10

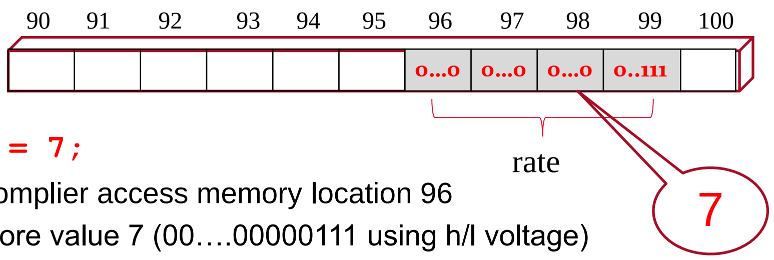
10





**`int rate;`**

- set aside memory (4 bytes)
- associates 96 (starting address) with **`rate`**;



**`rate = 7;`**

- Complier access memory location 96
- Store value 7 (00….00000111 using h/l voltage)
- Hidden from you

11

11

## C allows us to access and store the addresses of variables

Not in Java

**&x**

- address of a <u>variable</u>, <u>array element</u>. (No expression)

```
&x    &rate
&arr[0]; // later
scanf("%d %d", &a, &b);
```

**type * p ;**

- **p** is a pointer variable capable of storing the address of a int variable -- pointing to variable of type **type**

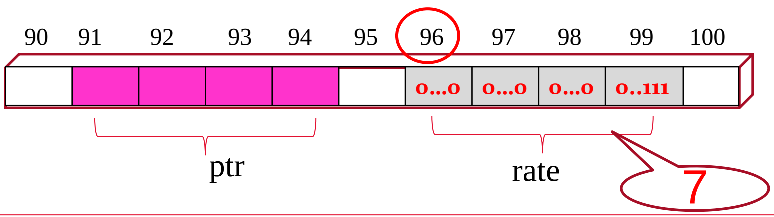
```
int * p, *q;
double * pd;
int j, a[10],  * p2, *q2;

p = &x;
int *r = &rate;
```

12

## Declare and initialize pointer

```
int *ptr;  /* declare a pointer to int  */
```

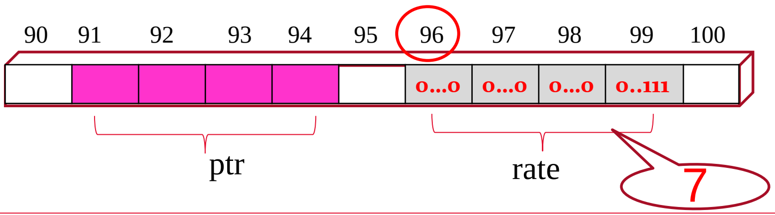
- Create a variable holding the address of other variable



## Declare and initialize pointer

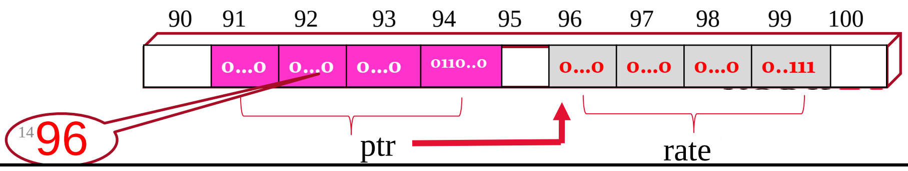
```
int *ptr;  /* declare a pointer to int  */
```

- Create a variable holding the address of other variable

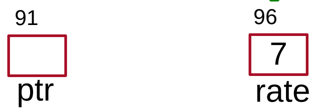


```
• ptr = &rate  /*assigning address of rate*/
```

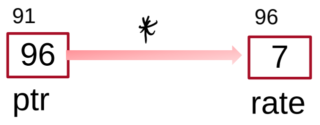
- Store address/pointer of **`rate`** in **`ptr`** (i.e., **`ptr'`**s value is the address)
- **`ptr`** now ‘points to’ **`rate`**



```
int *ptr;      /* I'm a pointer to an int */
```

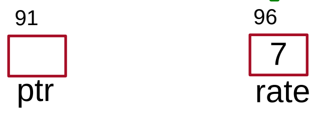


```
ptr = &rate;  /*I got the address of rate */
```

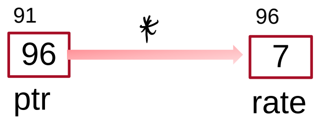


```
int *ptr;      /* I'm a pointer to an int */
```

mnemonic: "expression *ptr is an int"



```
ptr = &rate; /*I got the address of rate */
```



```
*ptr;      /* dereferencing. Indirect access.
              Get contents of the pointee */
```

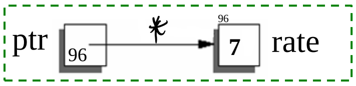
| | | |
|---|---|---|
| `ptr` | `&rate` | address of `rate` |
| `*ptr` | `rate` | content (value) of `rate` |

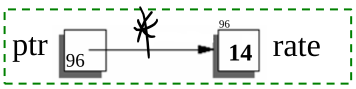
```
printf("%d", rate);    // 7  "direct access"
printf("%d", *ptr);    // 7  "indirect access"
```

17

17

```
int main()
{
  int rate = 7;
  int *ptr = &rate;
  printf("%d\n", rate); /* 7 */
  printf("%d\n", *ptr); /* 7 */

  int i = *ptr; // i=rate

  *ptr = 14; // rate = 14

  printf("%d %d\n", rate, *ptr); /* 14 14 */

 printf("%p %p\n", &rate, ptr); /*  96 96 */
}
```

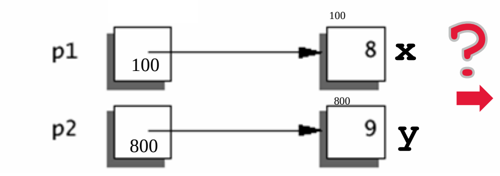




18

## Some example of Pointer basics

```
int *p1, *p2;  int x = 8, y = 9;
p1 = &x;  p2 = &y;
*p1 = *p2;    // x = y
```



Assume x is at address 100, y is at address 800

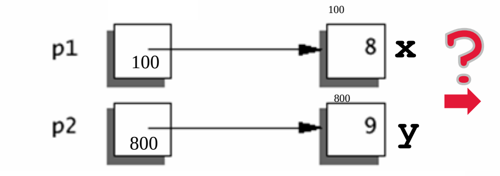
```
// copy value of p2's pointee (y) into pointee of p1 (x)
   *p1 is the alias of x    *p2 is the alias of y
```

19

19

## Some example of Pointer basics

```
int *p1, *p2;  int x = 8, y = 9;
p1 = &x;  p2 = &y;
p1 = p2;   /*copy the content of p2 (address of y) into p1
             now p1 also points to y  */
```



Assume x is at address 100, y is at address 800

Java:
```
Student s1 = new Student("John", 22);
Student s2 = new Student("Gorge",20);
s1 = s2;
```

RECALL

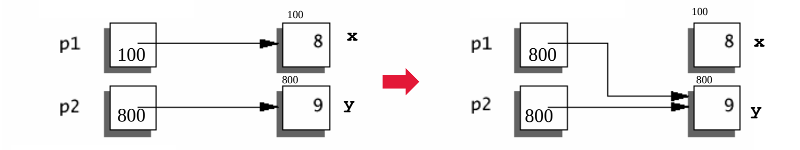
21

21

## Some example of Pointers -- summary

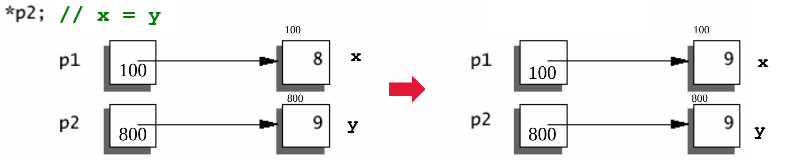
```
int *p1, *p2, x = 8, y = 9;
p1 = &x;  p2 = &y;
```

```
p1 = p2; // p1 = &y
```



```
printf("%d %d\n", *p1, *p2); // 9 9
printf("%p %p\n",  p1, p2); //  800 800
```

```
*p1 = *p2; // x = y
```



```
printf("%d %d\n", *p1, *p2); // 9 9
printf("%p %p\n",  p1, p2); //  100 800
```

Stopped here last time 23

23

## Precedence and Associativity  p53

| Operator Type | Operator | |
|---|---|---|
| Primary Expression Operators | () [] . -> | |
| Unary Operators | * & + - ! ~ ++ -- (typecast) sizeof | |
| Binary Operators | * / % | arithmetic |
| | + - | arithmetic |
| | >> << | bitwise |
| | < > <= >= | relational |
| | == != | relational |
| | & | bitwise |
| | ^ | bitwise |
| | \| | bitwise |
| | && | logical |
| | \|\| | logical |
| Ternary Operator | ?: | |
| Assignment Operators | = += -= *= /= %= >>= <<= &= ^= \|= | |
| Comma | , | |

```
ptr = &x;
*ptr = 5;

y= *ptr + 4

ptr= &arr[0]

No () needed here

But not always
*p.x = 5 (later)
```

24

| | | |
|---|---|---|
| `++ --` | Prefix increment/decrement | right-to-left |
| `+ -` | Unary plus/minus | |
| `! ~` | Logical negation/bitwise complement | |
| `(type)` | Cast (change *type*) | |
| `*` | Dereference | |
| `&` | Address | |
| `sizeof` | Determine size in bytes | |

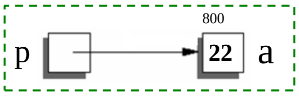
```
++ * ptr        * ptr;  * ptr = * ptr + 1
* ++ ptr        ptr = ptr +1;   *ptr;

(* ptr) ++      * ptr;  * ptr = * ptr + 1
* ptr ++        * ptr;  ptr = ptr +1
```

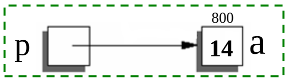
ptr +1 later

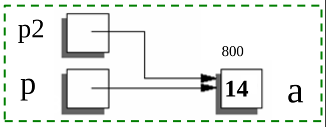
For your information

YORK U UNIVERSITÉ UNIVERSITY

```
int main()
{
  int a = 22;
  int *p = &a;
  printf("%d %d\n", a, *p);  /* 22 22 */

  *p =  14;  // a = 14
  printf("%d %d\n", a, *p);  /* 14 14 */

  int *p2 = p;

  (*p2)--;  // *p2 = *p2 - 1;
  printf("%d %d %d\n", a, *p, *p2);
  printf("%p %p %p\n", &a, p, p2);

  double d = 23.32;
  int *p3 = &d;  ???
  double * p3 = &a; ???
}
```



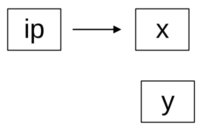




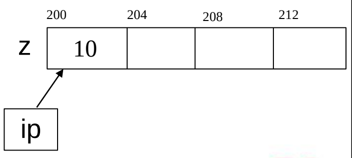
Not valid! Type must match

## Another example

```
int x = 1, y = 2, z[4], k;
int *ip;
ip = &x;          /* ip points to x */

y = *ip;          /* y = x   y is now 1 */
*ip = 0;          /* x is now 0, y? */
```



```
z[0] = 10;
ip = &z[0];       /* ip points to z[0] now */
for (k = 1; k < 4; k++)
  z[k] = *ip + k;

*ip += 100; // *ip = *ip + 100
            // z[0] = z[0]+100
(*ip)++;
```



27

```
x: 0  y: 1      z: 111 11 12 13
```

27

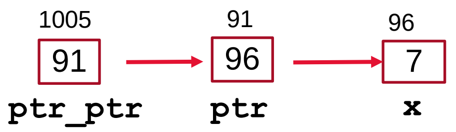
## Pointers K&R Ch 5

- Basics: Declaration and assignment (5.1)
- **Pointer to Pointer (5.6)**
- Pointer and functions (5.2)
- Pointer arithmetic (5.4)
- Pointers and arrays (5.3)
- Arrays of pointers (5.6)
- Command line argument (5.10)
- Pointer to arrays and two dimensional arrays (5.9)
- Pointer to functions (5.11)
- Pointer to structures (6.4)
- Memory allocation (extra)

Plan for today

28

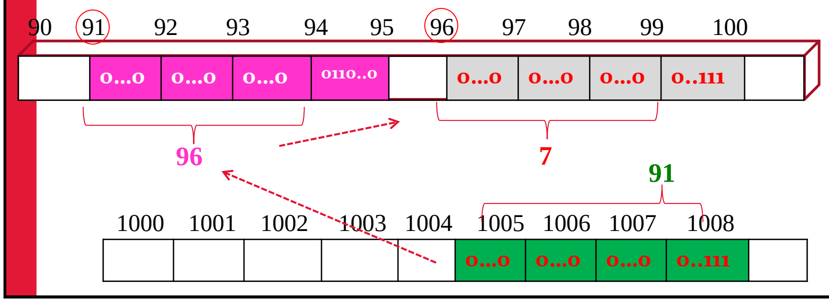
## Pointer to pointers



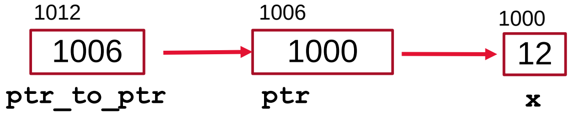
```
int x = 7;
int * ptr = &x;
```

```
int ** ptr_ptr          // a pointer to pointer
ptr_ptr = &ptr;         // ptr_ptr value is 91

** ptr_ptr = 20;        // ** access x, set x to 20
```

mnemonic:



## Pointer to pointers another example

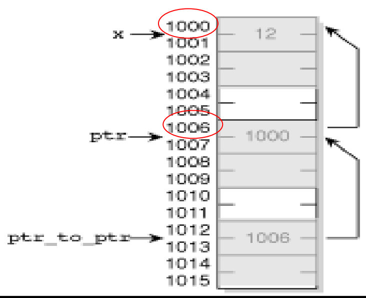


```
int x = 12;
int *ptr;
ptr = &x;
int **ptr_to_ptr          /* I am a pointer to pointer */
ptr_to_ptr = &ptr;        /* points to ptr */
**ptr_to_ptr = 20;        /* multiple indirection*/
```

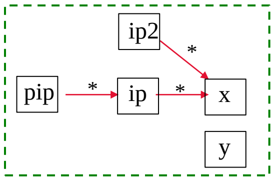
valid operations

| | | | |
|---|---|---|---|
| x, &x | | *x | ✗ |
| ptr &ptr | *ptr | **ptr | ✗ |
| ptr_to_ptr | &ptr_to_ptr | | |
| *ptr_to_ptr | **prt_to_ptr | | |

**ptr_to_ptr == *ptr == x;



## More Examples

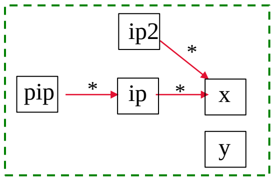


```
int x = 1, y = 2;
int *ip, *ip2;

ip = &x;

int **pip;      // I am a pointer to pointer
pip = &ip;      // pip points to pointer ip

y = **pip;
(**pip)--;

ip2 = ip;
*ip2 += 10;
```

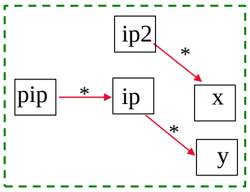
```
ip = &y;
(**pip)--;

printf("%d %d\n", x,  y);
```

31

## More Examples



```
int x = 1, y = 2;
int *ip, *ip2;

ip = &x;

int **pip;      // I am a pointer to pointer
pip = &ip;      // pip points to pointer ip

y = **pip;     // y=x  y is 1 now
(**pip)--;    // x=x-1  x is 0

ip2 = ip;
*ip2 += 10;  // *ip2=*ip2+10  x=x+10=10
```



```
ip = &y;
(**pip)--;  // y = y-1      y is 0 */

printf("%d %d\n", x,  y);   10  0
```

```
ip2 = pip; ???  Not valid! Type must match
pip = ip2; ???  Not valid! Type must match   y = *pip  ???
```

32

## Pointers  K&R Ch 5

- Basics: Declaration and assignment (5.1)
- Pointer to Pointer (5.6)
- **Pointer and functions (5.2)**
- Pointer arithmetic (5.4)
- Pointers and arrays (5.3)
- Arrays of pointers (5.6)
- Command line argument (5.10)
- Pointer to arrays and two dimensional arrays (5.9)
- Pointer to functions (5.11)
- Pointer to structures  (6.4)
- Memory allocation (extra)

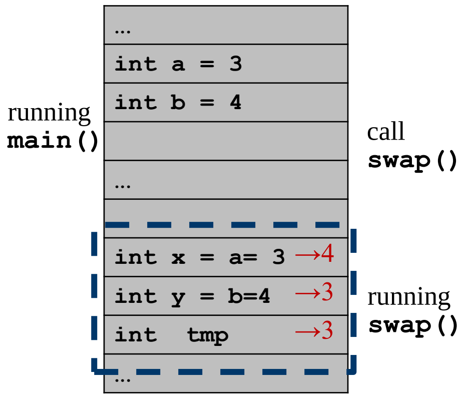
today

## Calling by Value

RECALL

- In C, all functions are **called by value**
  - Value of the arguments are passed to functions, but not the arguments themselves (i.e., not ~~call by reference~~)

```
void swap (int x, int y)
{
  int tmp;
  tmp = x;
  x = y;
  y = tmp;
}
main(){
  int a=3, b=4;
  swap(a,b);
```



*Send your friend attachment, for editing*

## Pointers and function arguments

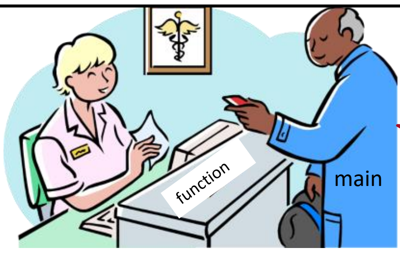
- In C, all functions are **called by value**
  - Value of the arguments are passed to functions, but not the arguments themselves (i.e., not ~~call by reference~~)
  - How to modify the arguments? **`increment()`** **`swap()`**
  - How to pass a structure such as array?

- Modify an actual argument by passing its address/pointer
  - Possibly modify passed arguments via their address!
  - Efficient.

35

*Send your friend a link to your file, instead of attachment, for editing 1)efficient, 2)can modify*

35



Mr. Main: Hi, function, I have some manuscripts, stored in lockers (memory), and I want you to repaint them so their color changed.

Ms function: Hi, Mr Main, here is how we function work: We don't take your original manuscript over the counter (not pass by reference). We always photocopy things (pass by value) passed here, and work on copies.

Mr. Main: Then, is there a way to have my original manuscripts' color changed?

Ms function: **Write down the locker number (address) on a paper, bring that paper to us (pass pointer/address) . We photocopy the paper (still pass by value), Then, based on the locker number on the copy, we go to your locker, fetch your original manuscripts there and work on them!**

36

36

## An example. Not working.

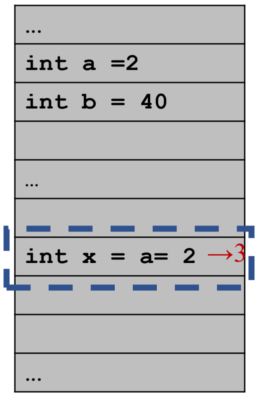
RECALL

```
void increment(int x)
{

   x++;

}

void main( ) {
   int a=2;

   increment(a);
   printf("%d", a);
}
```

Pass by value !!!

x = a

running
main()



2

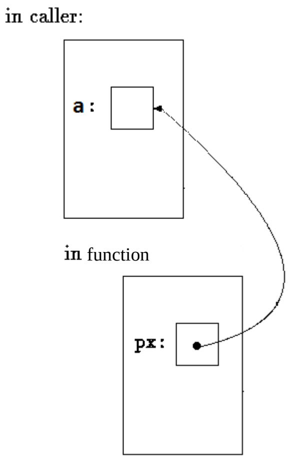
38

## The Correct Version

I am expecting int pointers

```
void increment(int *px)
{

                    ?

}

void main( ) {
   int a=2;

   increment(&a);
   printf("%d", a);
}
```

Pass address/pointer



39

## The Correct Version

I am expecting
int pointers

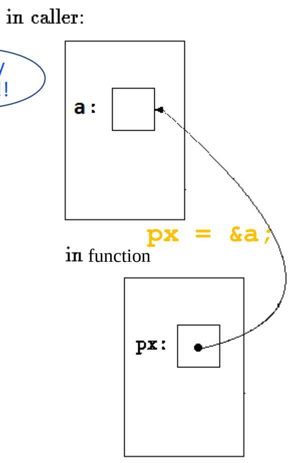
```
void increment(int *px)
{
   *px = *px + 1;  // *px is alias of a
              //  (*px) ++


}

void main( ) {
   int a=2;

   increment(&a);
   printf("%d", a);
}
```

px = &a

Pass by value !!!

3

Not in Java



40

## The Correct Version

I am expecting
int pointers

```
void increment(int *px)
{
   (*px) ++;



}

void main( ) {
   int a=2;
   int *pa = &a;
   increment(pa);
   printf("%d", a);
}
```

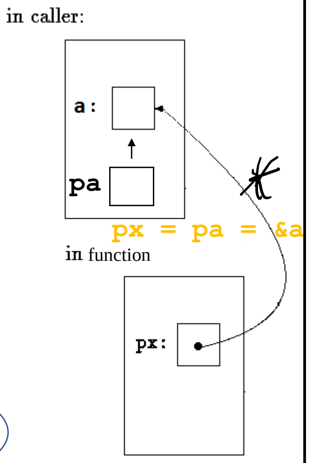
px = pa = &a

Pass by value !!!

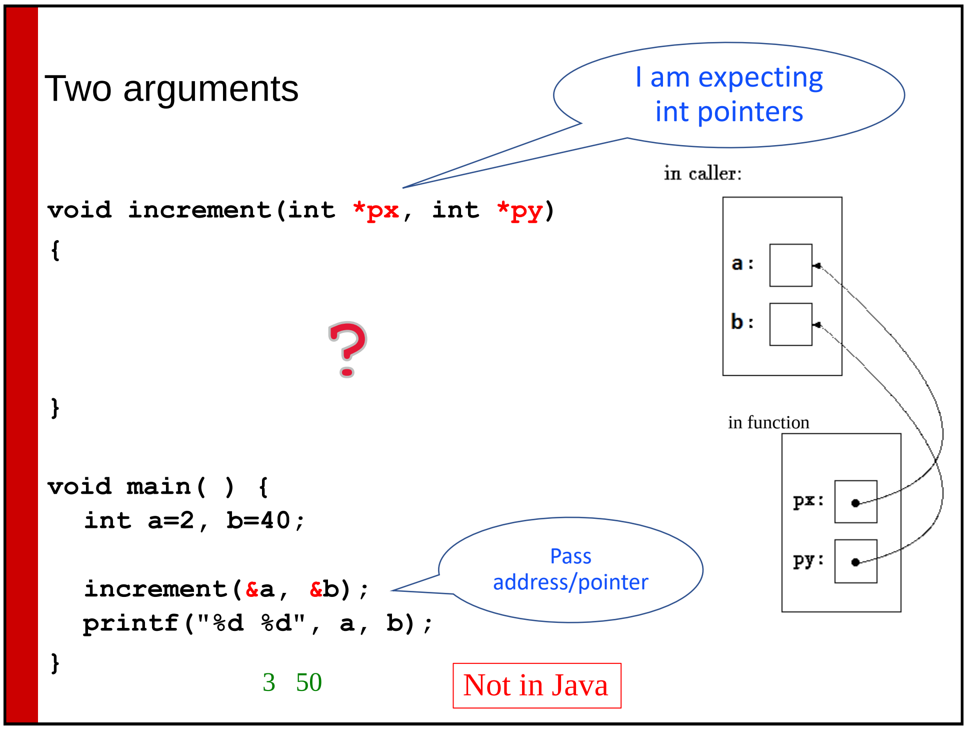
Pass address/pointer
Another way

3

Not in Java



41



## Two arguments

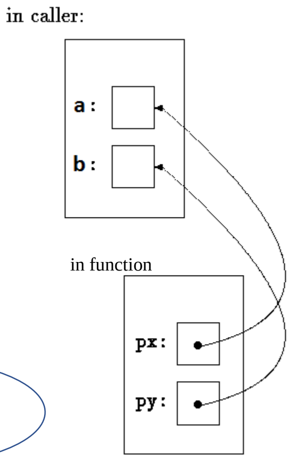
I am expecting int pointers

```
void increment(int *px, int *py)
{

            ?

}

void main( ) {
   int a=2, b=40;

   increment(&a, &b);
   printf("%d %d", a, b);
}
```

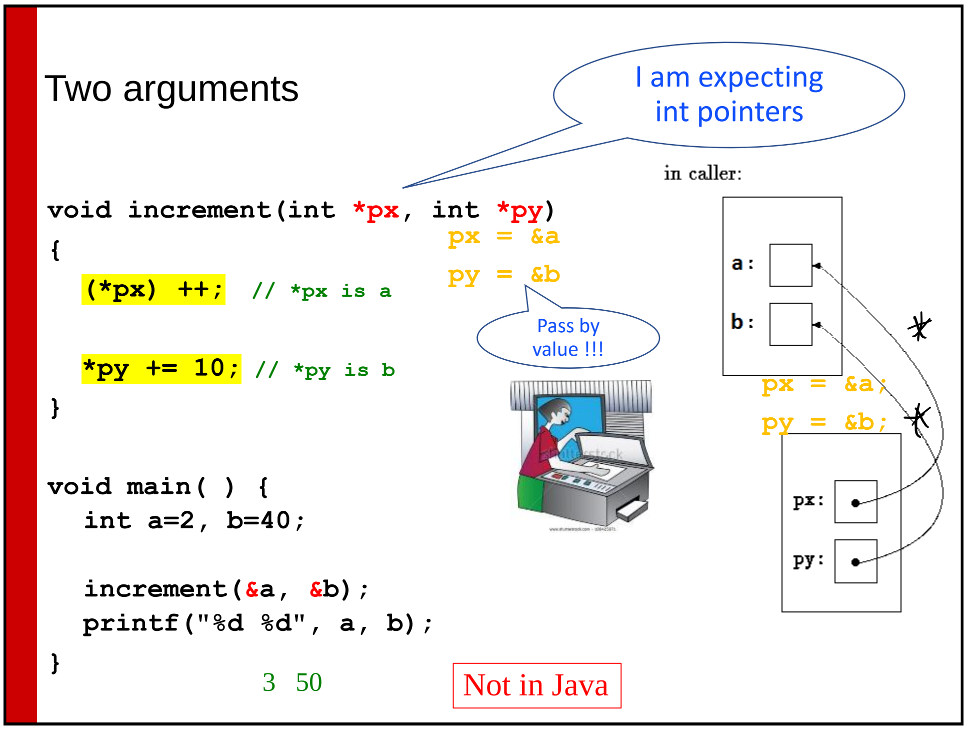
Pass address/pointer

3 50

Not in Java



42



## Two arguments

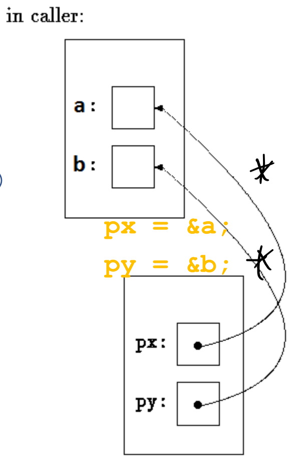
I am expecting int pointers

```
void increment(int *px, int *py)
{                                px = &a
   (*px) ++;  // *px is a        py = &b

   *py += 10; // *py is b
}

void main( ) {
   int a=2, b=40;

   increment(&a, &b);
   printf("%d %d", a, b);
}
```

Pass by value !!!

3 50

Not in Java



43

## Two arguments

I am expecting int pointers

```
void increment(int *px, int *py)
{                              px = pa = &a
   (*px) ++;                   py = pa = &b

   *py += 10;
}

void main( ) {
   int a=2, b=40;
   int *pa=&a;  int *pb=&b;
   increment(pa, pb);
   printf("%d %d", a, b);
}
```

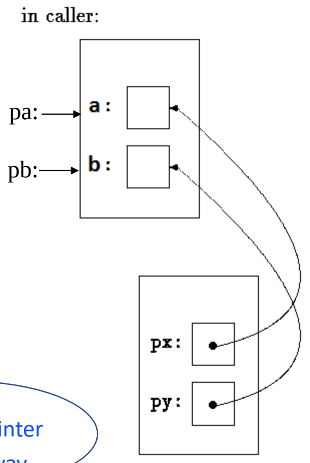
Pass by value !!!

Pass address/pointer

Another way

3 50

Not in Java



44

## Swap, the Correct Version

I am expecting int pointers

```
void swap(int *px, int *py)
{                              px = &a;
   int tmp;                    py = &b

}

void main( ) {
    int a=2, b=40;

    swap(&a, &b);

    printf("%d %d", a, b);
}
```

?

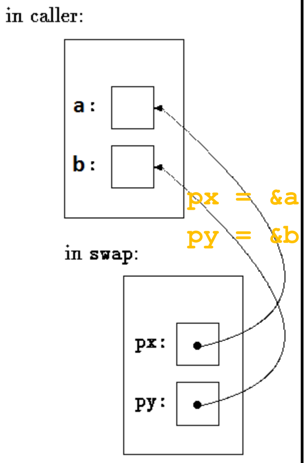
Pass by value !!!

Pass address/pointer

45

40 2

Not in Java



45

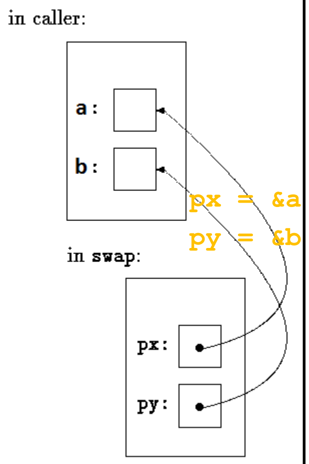
## Swap, the Correct Version

I am expecting int pointers

```
void swap(int *px, int *py)
{                              px = &a;
   int tmp;                    py = &b
   tmp = *px;   tmp=a;
   *px = *py;   a=b;
   *py = tmp;   b=tmp;
}

void main( ) {
    int a=2, b=40;

    swap(&a, &b);

    printf("%d %d", a, b);
}
```

Pass by value !!!

Pass address/pointer



Not in Java

46

40   2

46

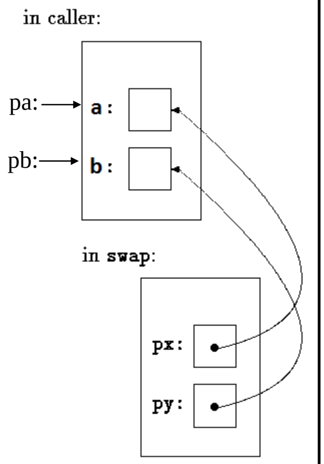
## Swap, the Correct Version

I am expecting int pointers

```
void swap(int *px, int *py)
{                          px = pa = &a;
   int tmp;                py = pb   = &b
   tmp = *px;
   *px = *py;
   *py = tmp;
}

void main( ) {
    int a=2, b=40;
    int *pa = &a;
    int *pb = &b;
    swap(pa,pb);
    printf("%d %d", a, b);
}
```

Pass by value !!!

Pass address/pointer, another way



We are not changing pointers

47

40   2

47

## Now understand scanf() -- more or less

```
int x=1;  int y = 2;
swap(&x,&y);  increment(&x,&y);
```

```
int x;
scanf ("%d", &x);
scanf ("%d  %d", &x, &y);
printf("%d", x);   // printf("%d", &x);
```

```
int x;
int *px = &x;
scanf("%d", px);
printf("%d",*px);
```

But why array name is used directly

```
scanf ("%d %s", &x, arrName)
fgets (arrName, 5,stdin);
```

explain shortly

48

## Another example

```
void swapIncre(int *px, int *py)
{
   int tmp;
   tmp = *px;
   *px = *py;
   *py = tmp;
    increment(? , ?);
}
```

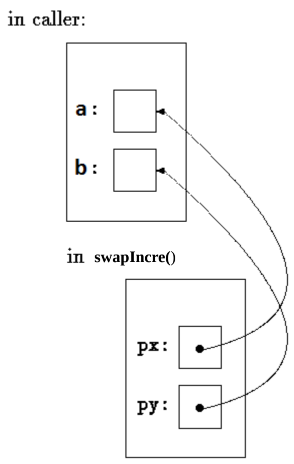
increment(px, py);

```
void increment(int *px2, int *py2)
{
    (*px2) ++;
    (*py2) += 10;
}
```

```
void main( ) {
    int a=2, b=40;

    swapIncre(&a, &b);
    printf("%d %d", a, b);
}
```

49

41  12



49

## Another example

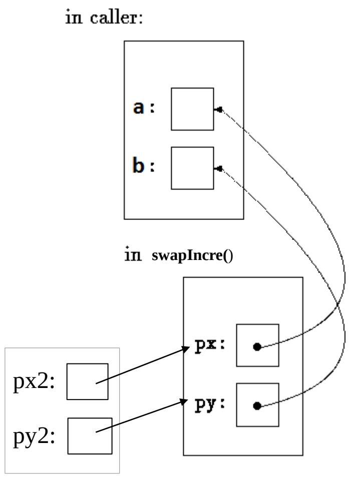
```
void swapIncre(int *px, int *py)
{
   int tmp;
   tmp = *px;
   *px = *py;
   *py = tmp;
    increment( &px , &py );
}
```

```
void increment(int **px2, int **py2)
{
    (**px2) ++;
    (**py2) += 10;
}
```

```
void main( ) {
    int a=2, b=40;

    swapIncre(&a, &b);
    printf("%d %d", a, b);
}
```

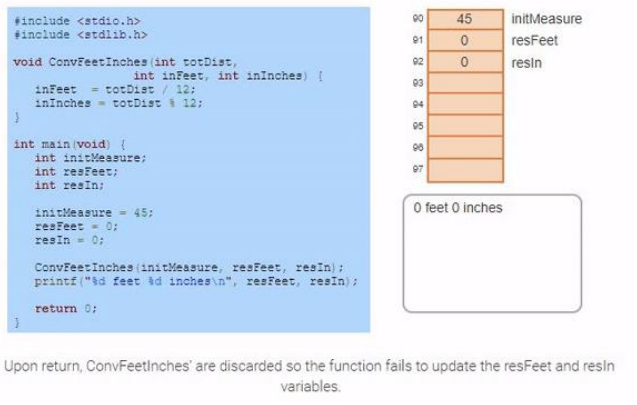
50

41  12



50

## Call by value



```
#include <stdio.h>
#include <stdlib.h>

void ConvFeetInches(int totDist,
                    int inFeet, int inInches) {
   inFeet   = totDist / 12;
   inInches = totDist % 12;
}

int main(void) {
   int initMeasure;
   int resFeet;
   int resIn;

   initMeasure = 45;
   resFeet = 0;
   resIn = 0;

   ConvFeetInches(initMeasure, resFeet, resIn);
   printf("%d feet %d inches\n", resFeet, resIn);

   return 0;
}
```

| | | |
|---|---|---|
| 90 | 45 | initMeasure |
| 91 | 0 | resFeet |
| 92 | 0 | resIn |
| 93 | | |
| 94 | | |
| 95 | | |
| 96 | | |
| 97 | | |

```
0 feet 0 inches
```

Upon return, ConvFeetInches' are discarded so the function fails to update the resFeet and resIn variables.

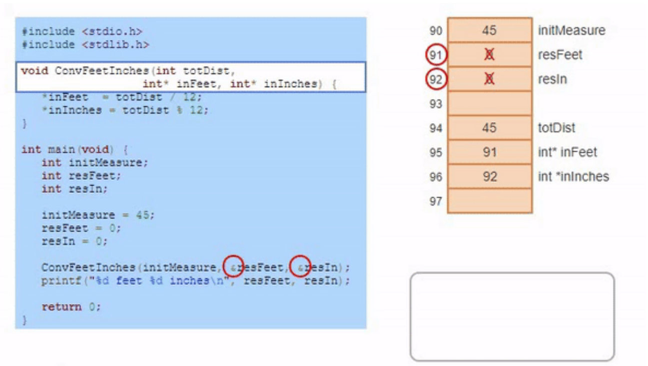
1. ConvFeetInches' parameters are passed by value, so the arguments' values are copied into local variables.
2. Upon return, ConvFeetInches' are discarded so the function fails to update the resFeet and resIn variables.

52

YORK U UNIVERSITÉ UNIVERSITY

52

## Call by value with pointers



```
#include <stdio.h>
#include <stdlib.h>

void ConvFeetInches(int totDist,
                    int* inFeet, int* inInches) {
   *inFeet   = totDist / 12;
   *inInches = totDist % 12;
}

int main(void) {
   int initMeasure;
   int resFeet;
   int resIn;

   initMeasure = 45;
   resFeet = 0;
   resIn = 0;

   ConvFeetInches(initMeasure, &resFeet, &resIn);
   printf("%d feet %d inches\n", resFeet, resIn);

   return 0;
}
```

| | | |
|---|---|---|
| 90 | 45 | initMeasure |
| 91 | X | resFeet |
| 92 | X | resIn |
| 93 | | |
| 94 | 45 | totDist |
| 95 | 91 | int* inFeet |
| 96 | 92 | int *inInches |
| 97 | | |

The & before the argument indicates that a variable's memory addresses, known as a pointer, is passed to a pass-by-pointer parameter. The * before the parameter name indicates the parameter is a pointer.

1. The & before the argument indicates that a variable's memory addresses, known as a pointer, is passed to a pass-by-pointer parameter. The * before the parameter name indicates the parameter is a pointer.
2. Prepending "*" to a pointer variable's name access the value pointed to by the pointer, so the original variable is updated.
3. Upon return from ConvFeetInches, resFeet and resIn retain their updated values, effectively "returning" two values.

53

53

## Pointers K&R Ch 5

- Basics: Declaration and assignment (5.1)
- Pointer to Pointer (5.6)
- Pointer and functions (5.2)

today

- Pointer arithmetic (5.4)
- Pointers and arrays (5.3)
- Arrays of pointers (5.6)
- Command line argument (5.10)
- Pointer to arrays and two dimensional arrays (5.9)
- Pointer to functions (5.11)
- Pointer to structures (6.4)
- Memory allocation (extra)

YORK U UNIVERSITÉ UNIVERSITY

54