Pointers K&R Ch 5

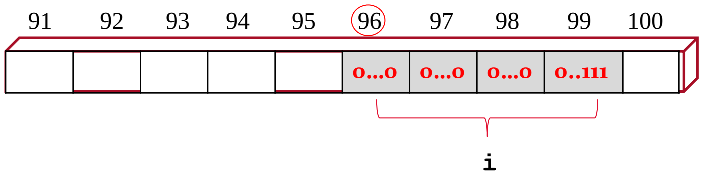
- Basics: Declaration and assignment (5.1)
- Pointer to Pointer (5.6)
- Pointer and functions (5.2)
- **Pointer arithmetic (5.4)**
- **Pointers and arrays (5.3)**

today

- 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)

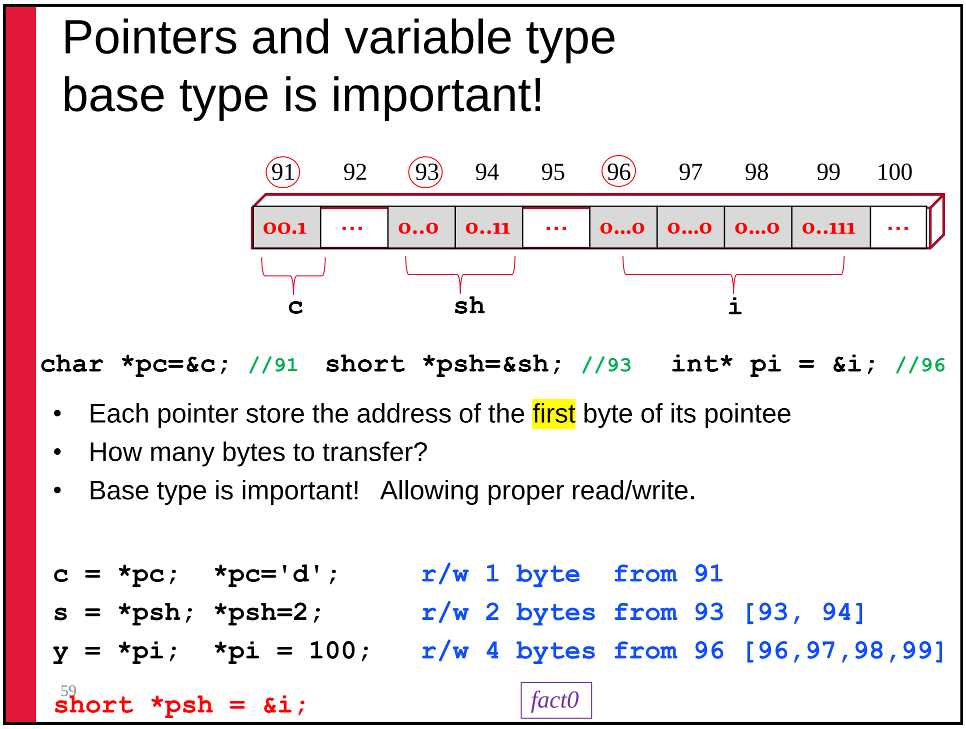
Pointers and variable type
base type is important!

```
int i = 7, y; int *pi;
pi = &i;   // pi stores 96, pointing to i
y = *pi;   // how many bytes to transfer? y = 7
```



- Each pointer stores the address of the first byte of its pointee
- How many bytes to transfer? -- Base type is important!

```
int i; char c; short sh;
int* pi; char *pc; short *psh;
```



Pointers and variable type
base type is important!

91 92 93 94 95 96 97 98 99 100

00.1 | … | 0..0 | 0..11 | … | 0…0 | 0…0 | 0…0 | 0..111 | …

c sh i

```
char *pc=&c; //91  short *psh=&sh; //93   int* pi = &i; //96
```

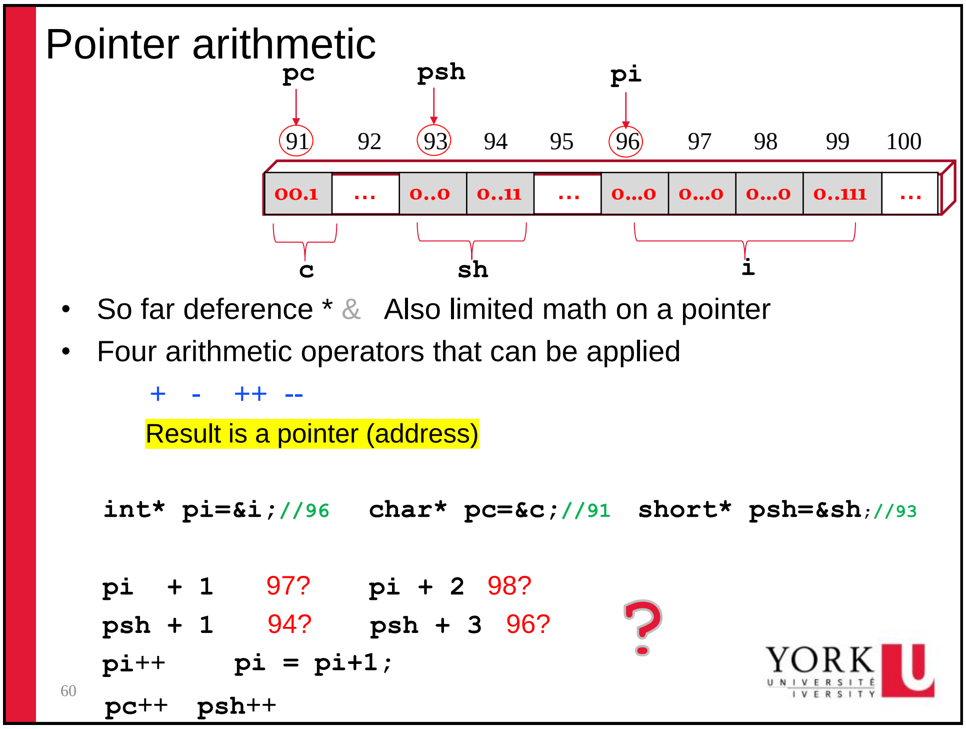
- Each pointer store the address of the first byte of its pointee
- How many bytes to transfer?
- Base type is important!   Allowing proper read/write.

```
c = *pc;   *pc='d';      r/w 1 byte  from 91
s = *psh; *psh=2;        r/w 2 bytes from 93 [93, 94]
y = *pi;   *pi = 100;    r/w 4 bytes from 96 [96,97,98,99]
```

59

```
short *psh = &i;
```

fact0



Pointer arithmetic

pc psh pi

91 92 93 94 95 96 97 98 99 100

00.1 | … | 0..0 | 0..11 | … | 0…0 | 0…0 | 0…0 | 0..111 | …

c sh i

- So far deference * & Also limited math on a pointer
- Four arithmetic operators that can be applied

  `+  -  ++ --`

  Result is a pointer (address)

```
int* pi=&i;//96   char* pc=&c;//91  short* psh=&sh;//93

pi  + 1     97?    pi + 2  98?
psh + 1     94?    psh + 3  96?
pi++      pi = pi+1;
pc++  psh++
```

?

YORK U UNIVERSITÉ UNIVERSITY

60

Pointer arithmetic – scaled

- Incrementing / decrementing a pointer by *n* moves it *n* units bytes
  p ± *n* ➔ p ± *n* × unit byte
  - value of a "unit" is based upon the size of the pointee type

fact1
    - p ± *n* ➔ p ± *n* × pointee-type-size

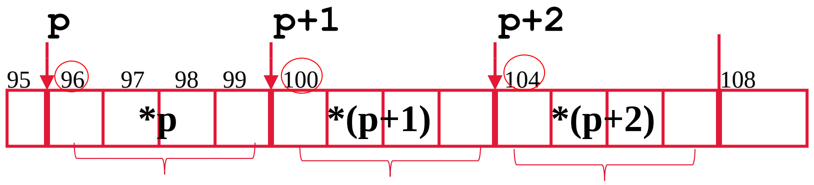
61

Pointer arithmetic – scaled

- Incrementing / decrementing a pointer by *n* moves it *n* units bytes
  p ± *n* ➔ p ± *n* × unit byte
  - value of a "unit" is based upon the size of the pointee type

fact1
    - p ± *n* ➔ p ± *n* × pointee-type-size

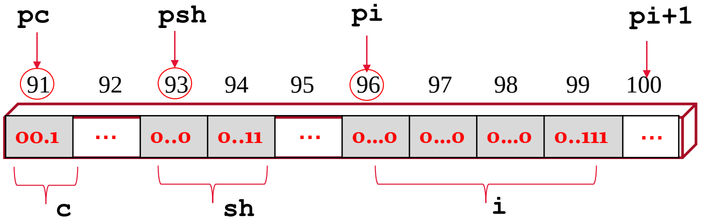
  - If p points to an integer (4 bytes), value of unit is 4
    p + n advances by n*4 bytes:
    p + 1 = 96 + 1*4 = 100   p + 2 = 96 + 2*4 = 104



- Why would we need to move pointer? p+1; p++
- Why designed this way? *"p+1 is p+4"*

62

Pointer arithmetic – scaled. *“p+1 is p+4”*



`int* pi=&i;//96`   `char *pc=&c;//91`   `short* psh=&sh;//93`

`pi + 1`   address 96+ 1*4 =100

`pi + 2`   address 96+ 2*4 =104

?

`psh +1`   address 93+ 1*2 =95

!!!

`psh +3`   address 93+ 3*2 =99   (other area)

assume
int   4 bytes
short 2 bytes
char 1 byte

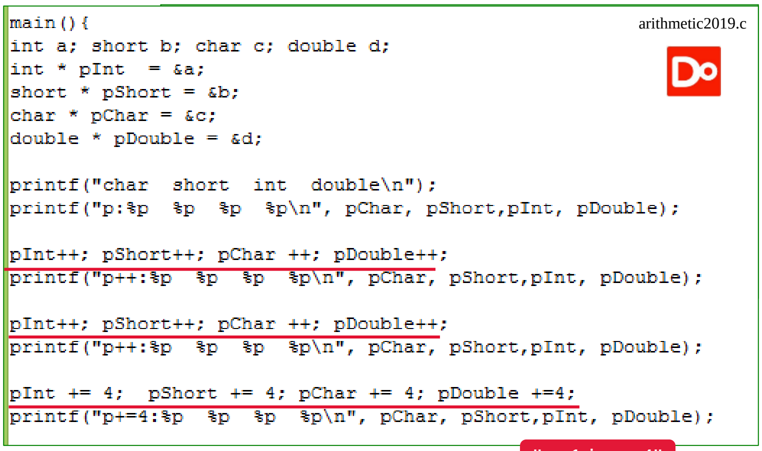
`pi++?`   `pc++?`   `psh++?`

63

`pi = 96 + 4`   `pc = 91 + 1`   `psh = 93 + 2`

YORK U UNIVERSITÉ UNIVERSITY

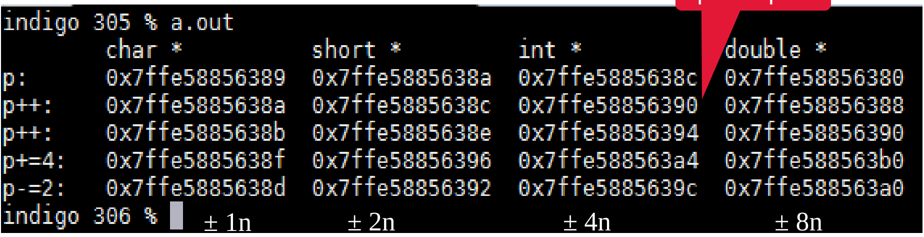
63



arithmetic2019.c

```
main(){
int a; short b; char c; double d;
int * pInt   = &a;
short * pShort = &b;
char * pChar = &c;
double * pDouble = &d;

printf("char  short  int  double\n");
printf("p:%p  %p  %p  %p\n", pChar, pShort,pInt, pDouble);

pInt++; pShort++; pChar ++; pDouble++;
printf("p++:%p  %p  %p  %p\n", pChar, pShort,pInt, pDouble);

pInt++; pShort++; pChar ++; pDouble++;
printf("p++:%p  %p  %p  %p\n", pChar, pShort,pInt, pDouble);

pInt += 4;  pShort += 4; pChar += 4; pDouble +=4;
printf("p+=4:%p  %p  %p  %p\n", pChar, pShort,pInt, pDouble);
```

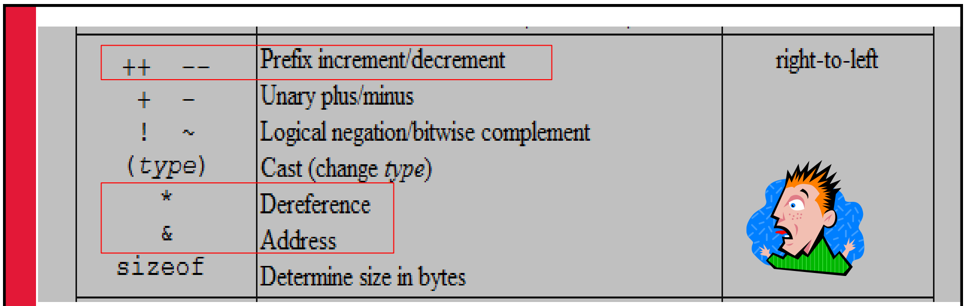
"p+1 is p+4"



```
indigo 305 % a.out
         char *          short *         int *           double *
p:       0x7ffe58856389  0x7ffe5885638a  0x7ffe5885638c  0x7ffe58856380
p++:     0x7ffe5885638a  0x7ffe5885638c  0x7ffe58856390  0x7ffe58856388
p++:     0x7ffe5885638b  0x7ffe5885638e  0x7ffe58856394  0x7ffe58856390
p+=4:    0x7ffe5885638f  0x7ffe58856396  0x7ffe588563a4  0x7ffe588563b0
p-=2:    0x7ffe5885638d  0x7ffe58856392  0x7ffe5885639c  0x7ffe588563a0
indigo 306 %
```

± 1n   ± 2n   ± 4n   ± 8n

64



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

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

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

For your information

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

Pointers K&R Ch 5

- Basics: Declaration and assignment (5.1)
- Pointer to Pointer (5.6)
- Pointer and functions (5.2) -- pass pointer by value
- Pointer arithmetic (5.4) +   - ++ --   “*p+1 is p+4*”

fact1

- **Pointers and arrays (5.3)**
  - **Arrays are stored consecutively** *fact2*
  - Pointer to array elements `p + i = &a[i]`   `*(p+i) = a[i]`
  - Array name contains address of 1st element   `a = &a[0]`
  - Pointer arithmetic on array (extension)
  - Array as function argument – “decay”
  - Pass sub_array

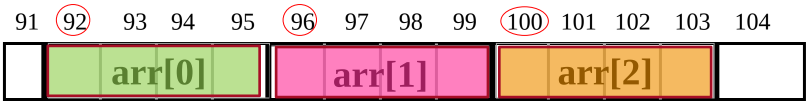
YORK U UNIVERSITÉ UNIVERSITY

Pointers and Arrays (5.3)

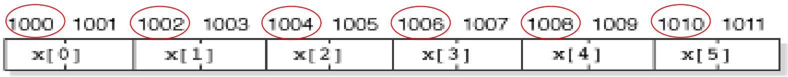
- Array members are next to each other in memory
  - `arr[0]` always occupies in the lowest address
  - `&arr[i+1]  ==  &arr[i]  +` size of type in byte`;`

fact2

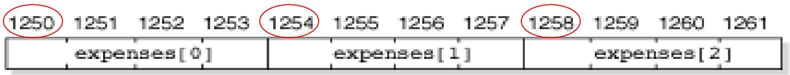
int arr[3]   size?



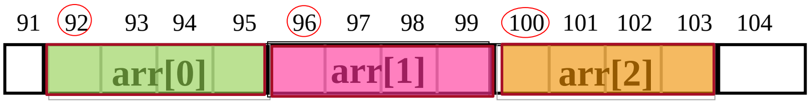
short `x[6];`   size?



`float expenses[3];`   size?



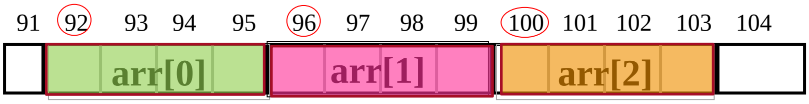
- Array members are next to each other in memory
  - **`arr[0]`** always occupies in the lowest address



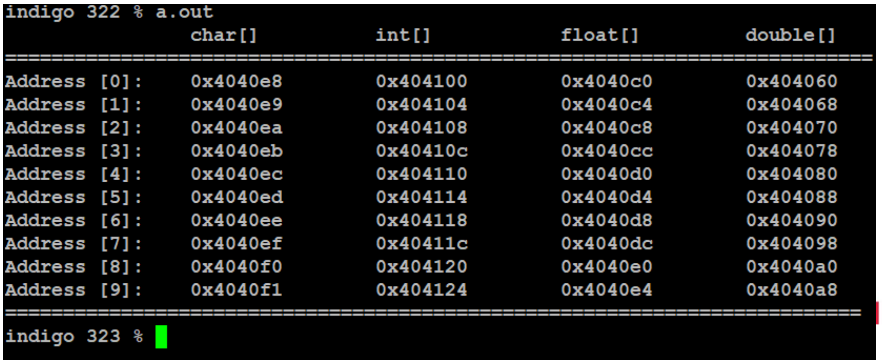
arrayElementSize2021.c

```
 int i[10], x;
 float f[10];
 double d[10];
 char c[10];

 main()
{

    /* Print the addresses of each array element. */
    printf("\n================================");

    for (x = 0; x < 10; x++)
        printf("\nElement [%d]: %p  %p %p %p", x, &c[x], &i[x],&f[x], &d[x]);

    printf("\n================================");
```

- Array members are next to each other in memory
  - **`arr[0]`** always occupies in the lowest address

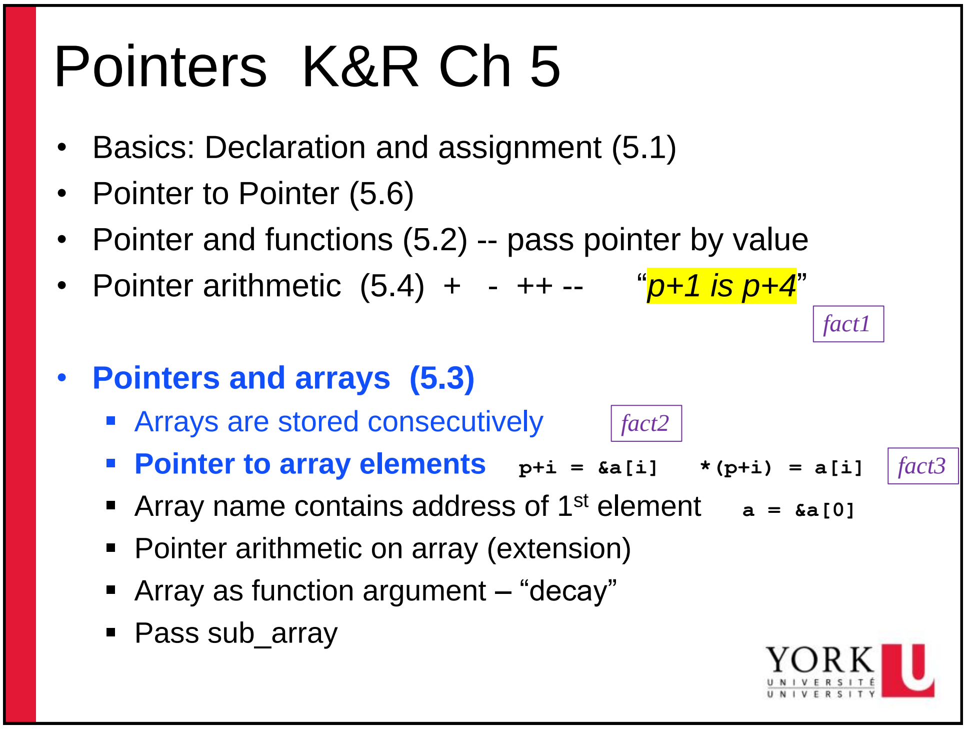


```
for (x = 0; x < 10; x++)

   printf("\nAddress [%d]: %p %p %p %p", x, &c[x], &i[x], &f[x], &d[x]);
```



```
indigo 322 % a.out
                 char[]           int[]            float[]          double[]
================================================================================
Address [0]:     0x4040e8         0x404100         0x4040c0         0x404060
Address [1]:     0x4040e9         0x404104         0x4040c4         0x404068
Address [2]:     0x4040ea         0x404108         0x4040c8         0x404070
Address [3]:     0x4040eb         0x40410c         0x4040cc         0x404078
Address [4]:     0x4040ec         0x404110         0x4040d0         0x404080
Address [5]:     0x4040ed         0x404114         0x4040d4         0x404088
Address [6]:     0x4040ee         0x404118         0x4040d8         0x404090
Address [7]:     0x4040ef         0x40411c         0x4040dc         0x404098
Address [8]:     0x4040f0         0x404120         0x4040e0         0x4040a0
Address [9]:     0x4040f1         0x404124         0x4040e4         0x4040a8
================================================================================
indigo 323 %
```

71 + 1 + 4 + 4 + 8

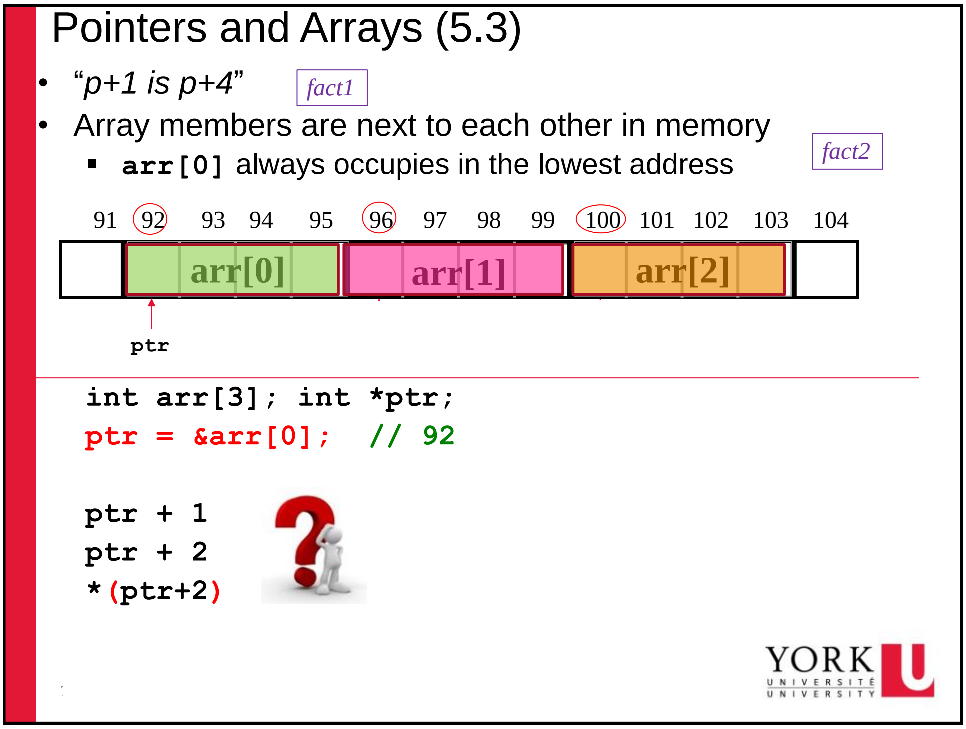


Pointers K&R Ch 5

- Basics: Declaration and assignment (5.1)
- Pointer to Pointer (5.6)
- Pointer and functions (5.2) -- pass pointer by value
- Pointer arithmetic (5.4) +   - ++ --   “*p+1 is p+4*”

fact1

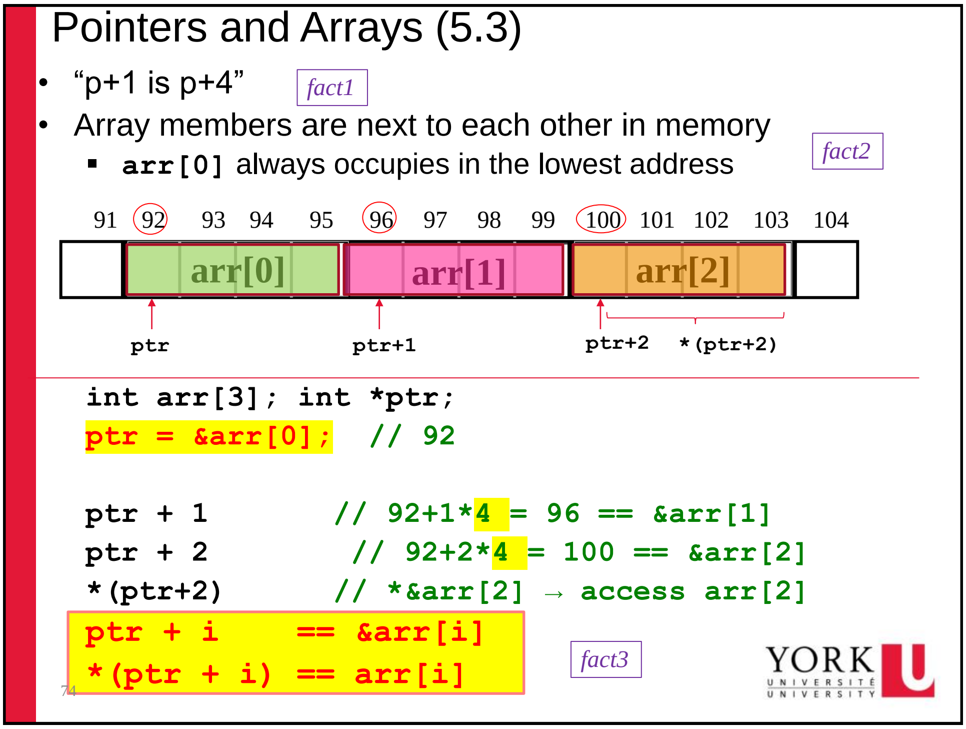
- **Pointers and arrays (5.3)**
  - Arrays are stored consecutively   *fact2*
  - **Pointer to array elements**   `p+i = &a[i]    *(p+i) = a[i]`   *fact3*
  - Array name contains address of 1st element   `a = &a[0]`
  - Pointer arithmetic on array (extension)
  - Array as function argument – “decay”
  - Pass sub_array



Pointers and Arrays (5.3)

- “*p+1 is p+4*”   *fact1*
- Array members are next to each other in memory   *fact2*
  - `arr[0]` always occupies in the lowest address

```
int arr[3]; int *ptr;
ptr = &arr[0];  // 92

ptr + 1
ptr + 2
*(ptr+2)
```



Pointers and Arrays (5.3)

- “p+1 is p+4” *fact1*
- Array members are next to each other in memory
  - **arr[0]** always occupies in the lowest address

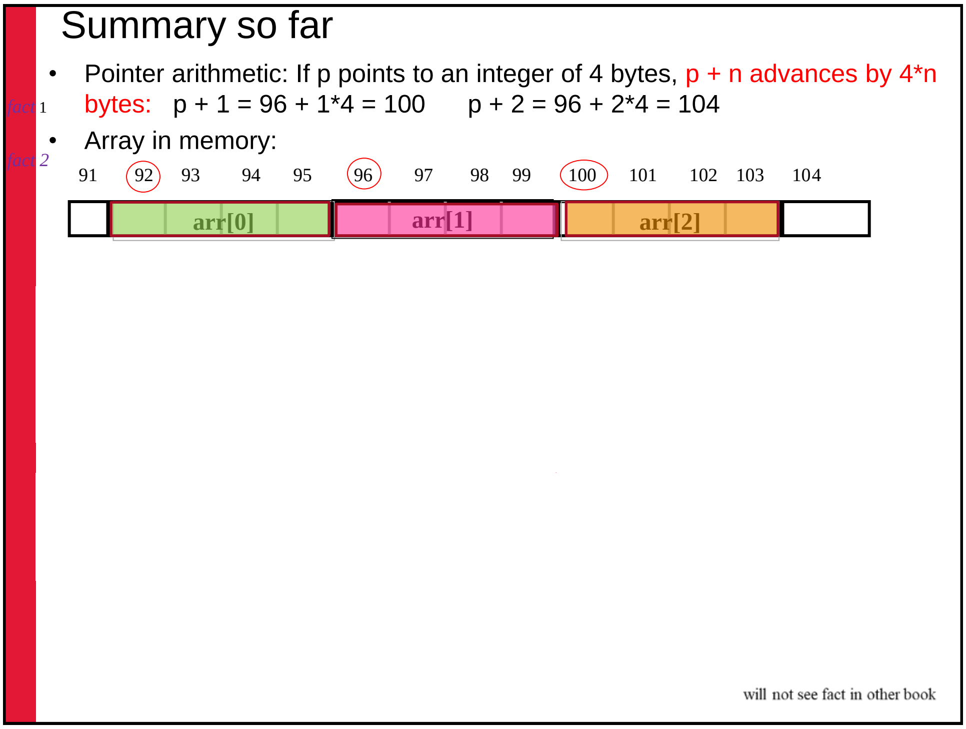
fact2

```
int arr[3]; int *ptr;
ptr = &arr[0];   // 92

ptr + 1         // 92+1*4 = 96 == &arr[1]
ptr + 2          // 92+2*4 = 100 == &arr[2]
*(ptr+2)        // *&arr[2] → access arr[2]
ptr + i      == &arr[i]
*(ptr + i)  == arr[i]
```

fact3

74

74



Summary so far

fact 1

- Pointer arithmetic: If p points to an integer of 4 bytes, p + n advances by 4*n bytes: p + 1 = 96 + 1*4 = 100   p + 2 = 96 + 2*4 = 104

fact 2

- Array in memory:

will not see fact in other book

75

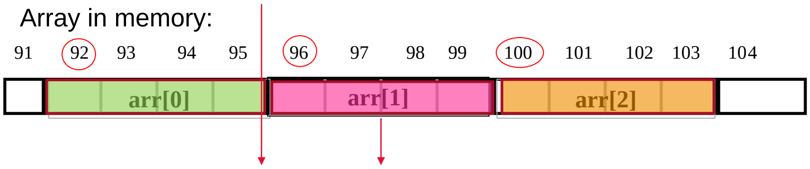
Summary so far

fact 1

- Pointer arithmetic: If p points to an integer of 4 bytes, p + n advances by 4*n bytes: p + 1 = 96 + 1*4 = 100   p + 2 = 96 + 2*4 = 104

fact 2

- Array in memory:



- Suppose `p` points to array element `k`, then `p+1` points to `k+1` (next) element. `p + i` points to `arr[k+i]`.
  - `p = &arr[k]:       p + i == &arr[k+i]   → *(p+i) == arr[k+i]`
  - `k=0:  p=&arr[0]:  p + i == &arr[i]   → *(p+i) == arr[i]`

fact 3

Pointers K&R Ch 5

- Basics: Declaration and assignment (5.1)
- Pointer to Pointer (5.6)
- Pointer and functions (5.2) -- pass pointer by value
- Pointer arithmetic (5.4) +   - ++ --   *"p+1 is p+4"*

fact1

- **Pointers and arrays (5.3)**
  - Arrays are stored consecutively   fact2
  - Pointer to array elements   **`p+i = &a[i]    *(p+i) = a[i]`**   fact3
  - **Array name contains address of 1st element**   fact4,5
  - Pointer arithmetic on array (extension)
  - Array as function argument – "decay"
  - Pass sub_array

YORK UNIVERSITÉ UNIVERSITY

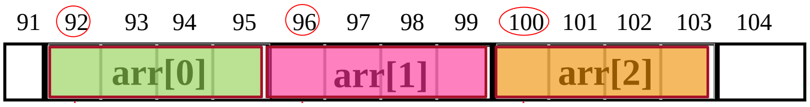
Pointers and Arrays (5.3)

- There is special relationship between pointers and arrays
- When you use array, you are using pointers!

```
  int i, arr[20], char c;
  scanf("%d  %c  %s", &i, &c, arr); // &arr is wrong
```

- Identifier (name) of an array is equivalent to the address of its 1st element. **arr == &arr[0]** *fact4*
  - Array name can be used ‘like’ a pointer. Follow pointer arithmetic rule!

```
arr + 1?
arr + 2?
```



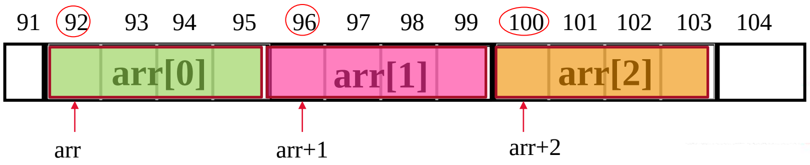
Pointers and Arrays (5.3)

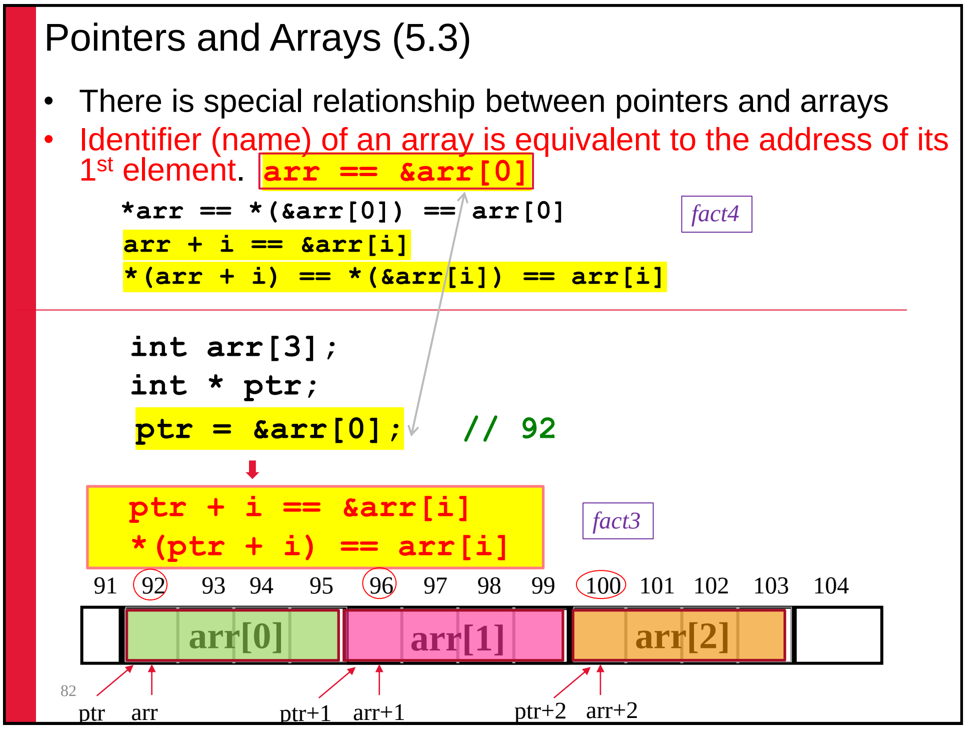
- There is special relationship between pointers and arrays
- When you use array, you are using pointers!

```
  int i, arr[20], char c;
  scanf("%d  %c  %s", &i, &c, arr); // &arr is wrong
```

- Identifier (name) of an array is equivalent to the address of its 1st element. **arr == &arr[0]** *fact4*
  - Array name can be used ‘like’ a pointer. Follow pointer arithmetic rule!

```
arr + 1? 92+4  == address of next element == &arr[1]
arr + 2? 92+8  == &arr[2]
*(arr + 2)?  == *(&arr[2]) == arr[2]
```





Pointers and Arrays (5.3)

- There is special relationship between pointers and arrays
- Identifier (name) of an array is equivalent to the address of its 1st element. `arr == &arr[0]`

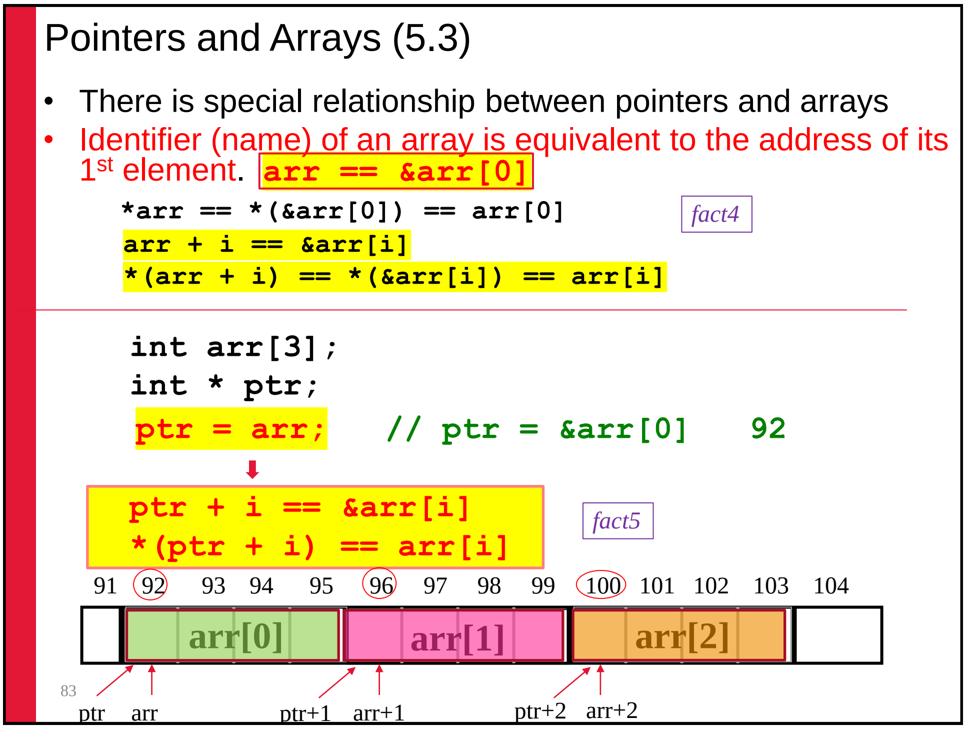
```
*arr == *(&arr[0]) == arr[0]
arr + i == &arr[i]
*(arr + i) == *(&arr[i]) == arr[i]
```

fact4

```
int arr[3];
int * ptr;
ptr = &arr[0];    // 92
```

```
ptr + i == &arr[i]
*(ptr + i) == arr[i]
```

fact3



Pointers and Arrays (5.3)

- There is special relationship between pointers and arrays
- Identifier (name) of an array is equivalent to the address of its 1st element. `arr == &arr[0]`

```
*arr == *(&arr[0]) == arr[0]
arr + i == &arr[i]
*(arr + i) == *(&arr[i]) == arr[i]
```

fact4

```
int arr[3];
int * ptr;
ptr = arr;    // ptr = &arr[0]    92
```

```
ptr + i == &arr[i]
*(ptr + i) == arr[i]
```

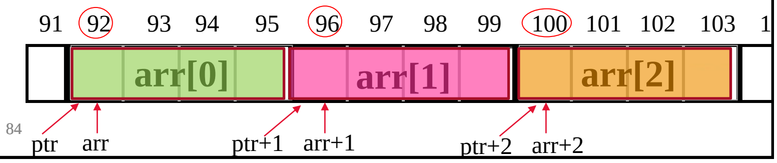
fact5

- There is special relationship between pointers and arrays
- Identifier (name) of an array is equivalent to the address of its 1st element. `arr == &arr[0]`

```
int arr[3]; int * ptr;
ptr = arr;   /*   ptr = &arr[0]   */
```

fact,4, 5

```
arr+i == &arr[i]
ptr+i == &arr[i]
```



84

- There is special relationship between pointers and arrays
- Identifier (name) of an array is equivalent to the address of its 1st element. `arr == &arr[0]`

```
int arr[3]; int * ptr;
ptr = arr;   /*   ptr = &arr[0]   */
```

fact,4, 5

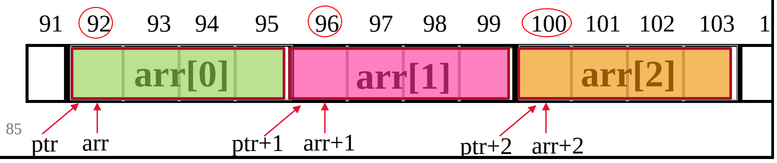
```
arr+i == &arr[i]
ptr+i == &arr[i]
```

➡

```
arr[i]

*(ptr + i)

*(arr + i)

ptr[i];
```

equivalent

`Compiler converts arr[2] to *(arr+2)`



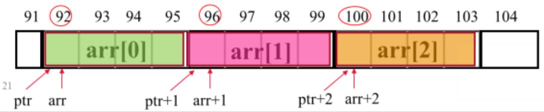
85

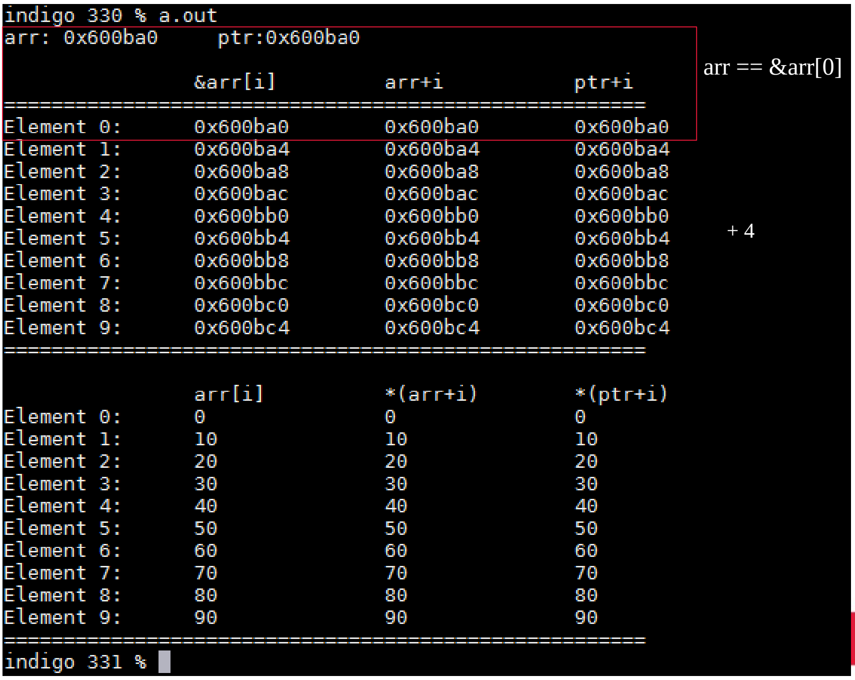
arrayAddressPPP.c

```
/* Demonstrates use of pointer arithmetic in array*/
main() {
 int arr[10] = {0,10,20,30,40,50,60,70,80,90}, i;
 int *ptr = arr;    /* = &arr[0] */

 printf("%p %p", arr, ptr); // print array name!

/* Print the addresses of each array element. */
 for (i = 0; i < 10; i++)
   printf("%p %p %p", &arr[i], arr+i, ptr+i);

/* Print the content of each array element. */
 for (i = 0; i < 10; i++)
   printf("%d %d %d", arr[i], *(arr+i), *(ptr+i));

}
```

Different ways of accessing array element addresses

Different ways of accessing array elements

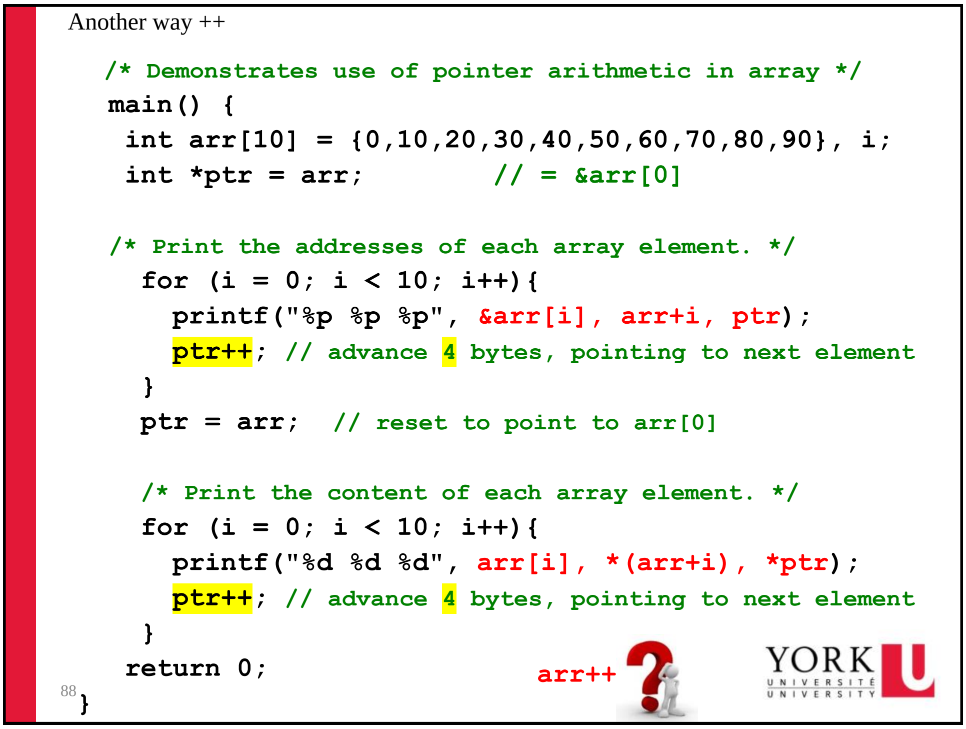




```
indigo 330 % a.out
arr: 0x600ba0     ptr:0x600ba0

                &arr[i]         arr+i           ptr+i
======================================================
Element 0:      0x600ba0        0x600ba0        0x600ba0
Element 1:      0x600ba4        0x600ba4        0x600ba4
Element 2:      0x600ba8        0x600ba8        0x600ba8
Element 3:      0x600bac        0x600bac        0x600bac
Element 4:      0x600bb0        0x600bb0        0x600bb0
Element 5:      0x600bb4        0x600bb4        0x600bb4
Element 6:      0x600bb8        0x600bb8        0x600bb8
Element 7:      0x600bbc        0x600bbc        0x600bbc
Element 8:      0x600bc0        0x600bc0        0x600bc0
Element 9:      0x600bc4        0x600bc4        0x600bc4
======================================================

                arr[i]          *(arr+i)        *(ptr+i)
Element 0:      0               0               0
Element 1:      10              10              10
Element 2:      20              20              20
Element 3:      30              30              30
Element 4:      40              40              40
Element 5:      50              50              50
Element 6:      60              60              60
Element 7:      70              70              70
Element 8:      80              80              80
Element 9:      90              90              90
======================================================
indigo 331 %
```

arr == &arr[0]

+ 4



Another way ++

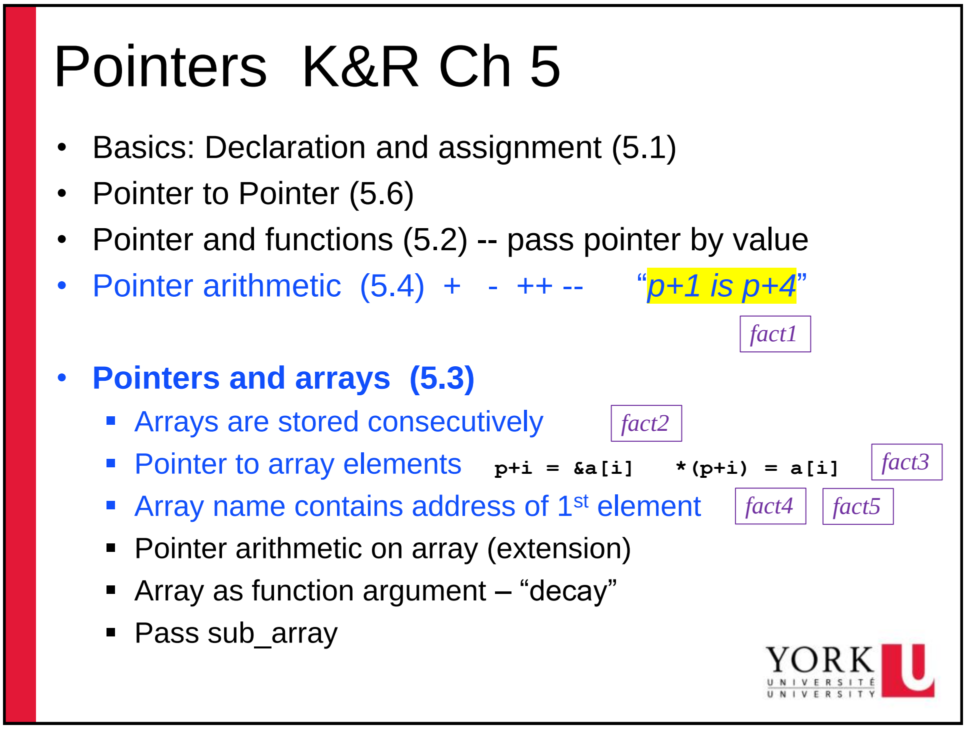
```
/* Demonstrates use of pointer arithmetic in array */
main() {
  int arr[10] = {0,10,20,30,40,50,60,70,80,90}, i;
  int *ptr = arr;          // = &arr[0]

 /* Print the addresses of each array element. */
    for (i = 0; i < 10; i++){
      printf("%p %p %p", &arr[i], arr+i, ptr);
      ptr++; // advance 4 bytes, pointing to next element
    }
    ptr = arr;  // reset to point to arr[0]

    /* Print the content of each array element. */
    for (i = 0; i < 10; i++){
      printf("%d %d %d", arr[i], *(arr+i), *ptr);
      ptr++; // advance 4 bytes, pointing to next element
    }
  return 0;
}
```

arr++ ?

YORK U UNIVERSITÉ UNIVERSITY

88

88



Pointers K&R Ch 5

- Basics: Declaration and assignment (5.1)
- Pointer to Pointer (5.6)
- Pointer and functions (5.2) -- pass pointer by value
- Pointer arithmetic (5.4) + - ++ --   "p+1 is p+4"

  fact1

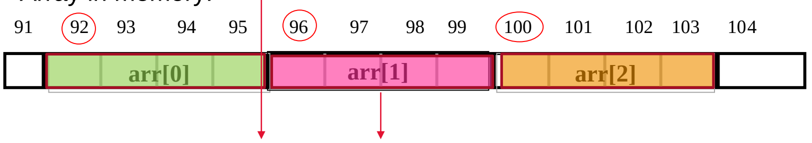
- **Pointers and arrays (5.3)**
  - Arrays are stored consecutively *fact2*
  - Pointer to array elements `p+i = &a[i]   *(p+i) = a[i]` *fact3*
  - Array name contains address of 1st element *fact4* *fact5*
  - Pointer arithmetic on array (extension)
  - Array as function argument – "decay"
  - Pass sub_array

YORK U UNIVERSITÉ UNIVERSITY

89

Summary

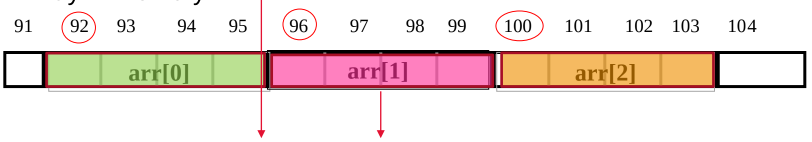
- fact 1 Pointer arithmetic: If p points to an integer of 4 bytes, p + n advances by 4*n bytes: p + 1 = 96 + 1*4 = 100 p + 2 = 96 + 2*4 = 104
- fact 2 Array in memory:



- Suppose `p` points to array element `k`, then `p+1` points to `k+1` (next) element. `p + i` points to `arr[k+i]`.
  - fact 3 `p = &arr[k]:      p + i == &arr[k+i]   → *(p+i) == arr[k+i]`
  - `k=0: p=&arr[0]:  p + i == &arr[i]    → *(p+i) == arr[i]`

Summary

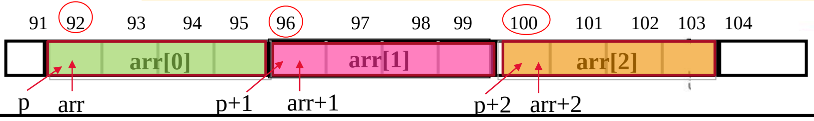
- fact 1 Pointer arithmetic: If p points to an integer of 4 bytes, p + n advances by 4*n bytes: p + 1 = 96 + 1*4 = 100 p + 2 = 96 + 2*4 = 104
- fact 2 Array in memory:

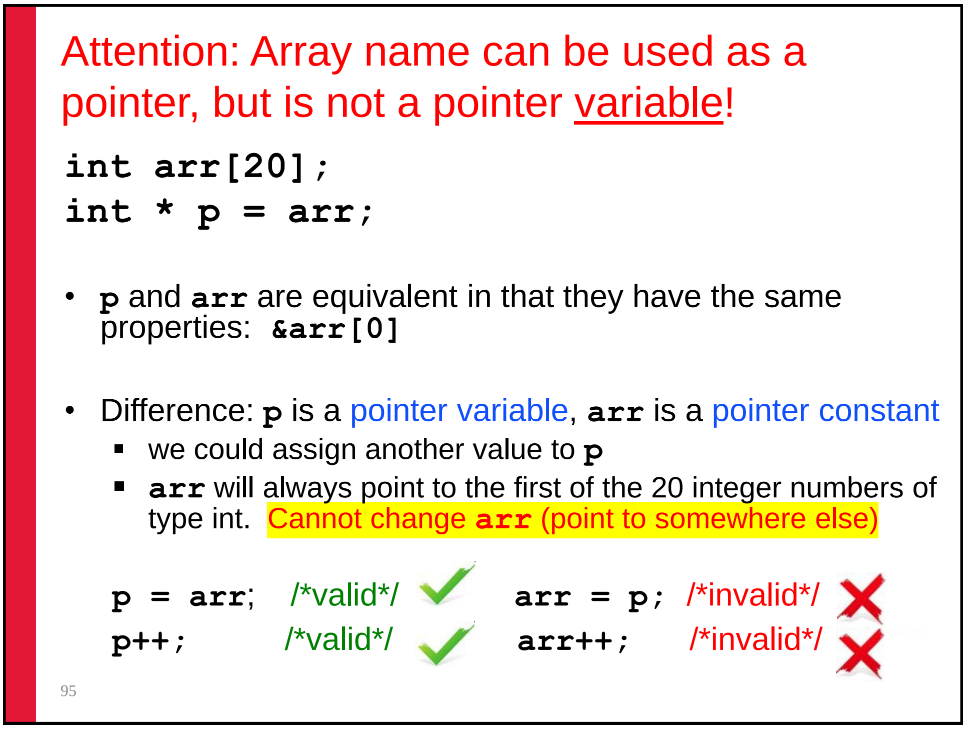


- Suppose `p` points to array element `k`, then `p+1` points to `k+1` (next) element. `p + i` points to `arr[k+i]`.
  - fact 3 `p = &arr[k]:      p + i == &arr[k+i]   → *(p+i) == arr[k+i]`
  - `k=0: p=&arr[0]:  p + i == &arr[i]    → *(p+i) == arr[i]`

- **Array name contains pointer to 1st element** `arr==&arr[0]`
  - fact 4 `arr==&arr[0]:    arr+i == &arr[i]   → *(arr+i) == arr[i]`

fact 5 **`p = arr:    p + i == &arr[i]  →   *(p+i) == arr[i]`**



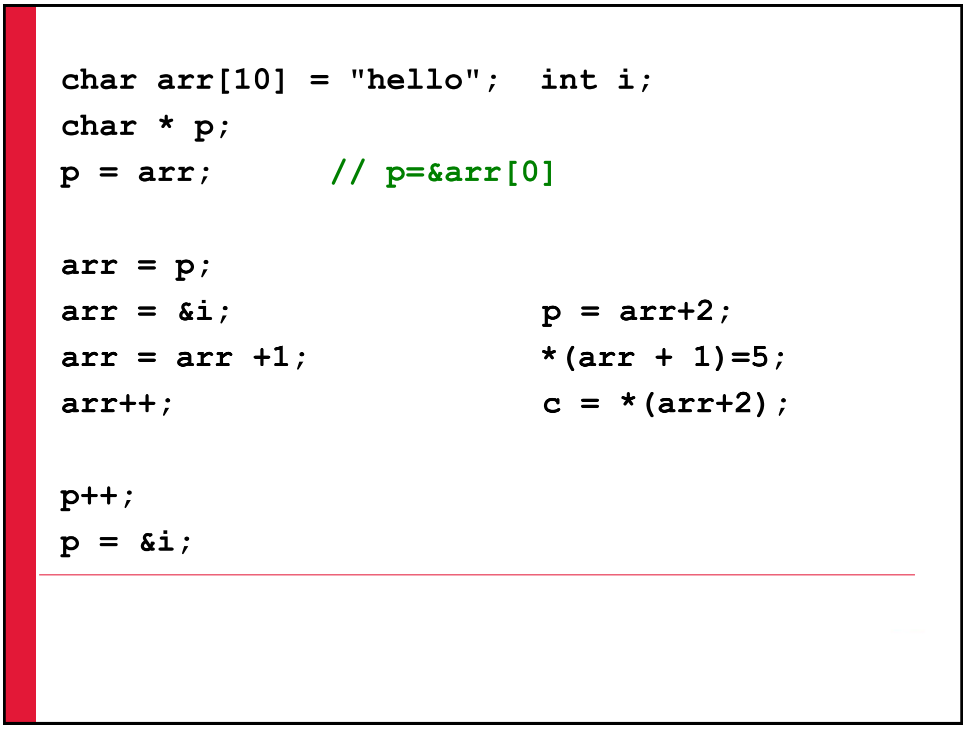


Attention: Array name can be used as a pointer, but is not a pointer variable!

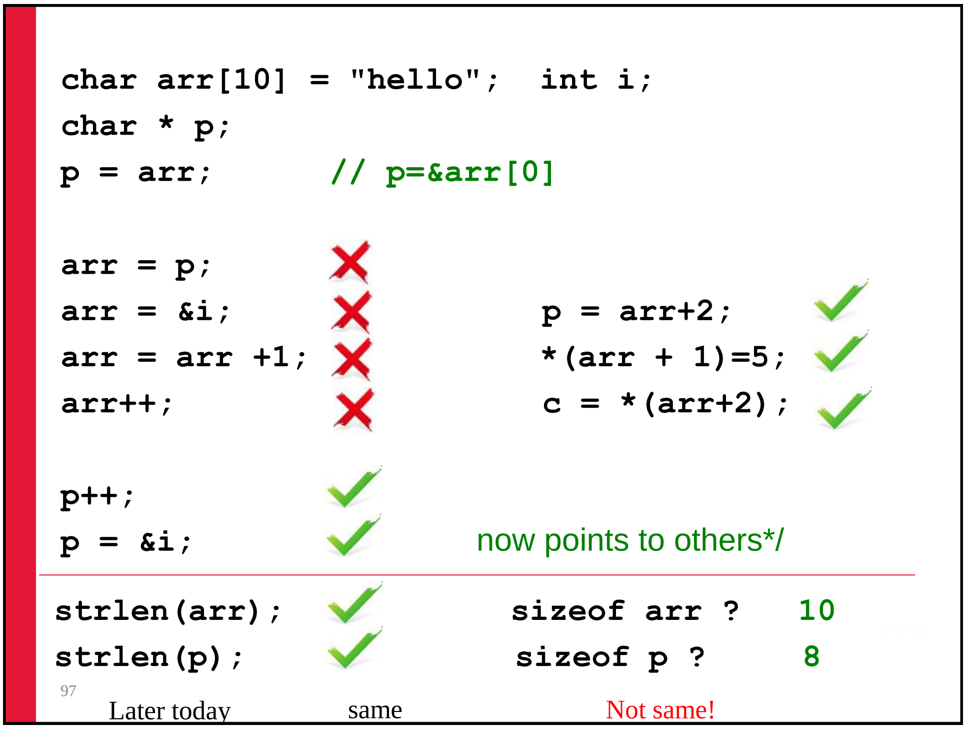
```
int arr[20];
int * p = arr;
```

- **`p`** and **`arr`** are equivalent in that they have the same properties: **`&arr[0]`**
- Difference: **`p`** is a pointer variable, **`arr`** is a pointer constant
  - we could assign another value to **`p`**
  - **`arr`** will always point to the first of the 20 integer numbers of type int. Cannot change **`arr`** (point to somewhere else)

```
p = arr;   /*valid*/ ✓      arr = p;  /*invalid*/ ✗
p++;       /*valid*/ ✓      arr++;    /*invalid*/ ✗
```



```
char arr[10] = "hello";  int i;
char * p;
p = arr;       // p=&arr[0]

arr = p;
arr = &i;                    p = arr+2;
arr = arr +1;                *(arr + 1)=5;
arr++;                       c = *(arr+2);

p++;
p = &i;
```



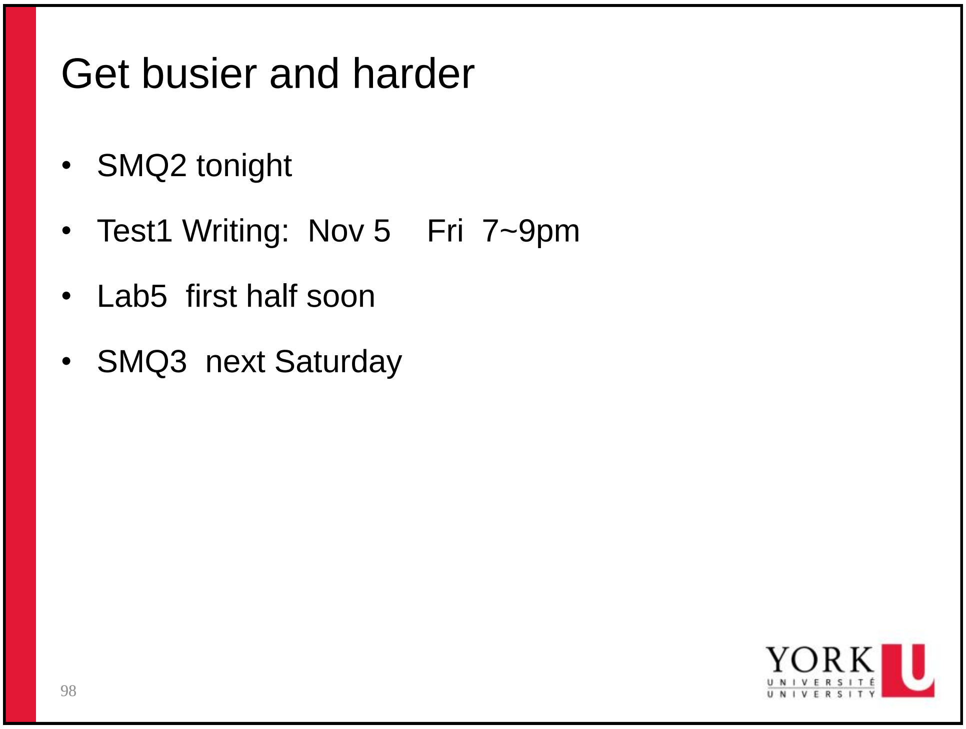
```
char arr[10] = "hello";  int i;
char * p;
p = arr;      // p=&arr[0]

arr = p;      ✗
arr = &i;     ✗          p = arr+2;       ✓
arr = arr +1; ✗          *(arr + 1)=5;    ✓
arr++;        ✗          c = *(arr+2);    ✓

p++;          ✓
p = &i;       ✓          now points to others*/
```

```
strlen(arr);  ✓          sizeof arr ?   10
strlen(p);    ✓          sizeof p ?      8
```

97

Later today same Not same!

97



Get busier and harder

- SMQ2 tonight
- Test1 Writing: Nov 5   Fri 7~9pm
- Lab5 first half soon
- SMQ3 next Saturday

98

98