1

- Midterm next week. “Midterm” page up shortly.
- Lab 5 posted soon this week.
  - Due after midterm.
  - Good practice.

2

YORK UNIVERSITÉ UNIVERSITY

2

```
printf("input name age and wage: ");
fgets(inputs[count], 50, stdin);   /* read in directly. add a \n */
sscanf(inputs[count],"%s %d %f %c", name, &age, &rate, &c);

while(! strcmp(inputs[count], "quit\n")){
   sprintf(inputs[count],"%s %d %.1f\n",name,age,rate,c); //redundant
   age += 10;  wage *= 1.5;
   name2[30];
   for(i=0; i< strlen(name); i++)
     name2[i] = name[i] + 1; // convert to next character
   name2[i] = '\0';  //needed!!
   ….
   char maxC = name[0];
   for(i=1; i< strlen(name); i++)
      if (name[i] > maxC) // has higher encoding in ASCII
        maxC = name[i];

   sprintf(inputs[count+1],"%s-%d-%.3f-[%d,.0f]-(%d)-'%c'\n",
          name2,age,rate, (int)floor(rate), ceil(rate), index, maxC);
   count += 2;
   fgets(inputs[count], 50, stdin); /* add a \n */
   sscanf(inputs[count],"%s %d %f", name, &age, &rate, &c);
}
```

Either way

3

	Non-string (int, char, float…)	String
Read scanf, sscanf fgets…	**&x**	**arrName**
Write printf, sprint puts…	**x**	**arrName**

- `scanf("%s %d %f %c", name, &age, &rate, &c);`
- `sscanf(table[i], "%s %d %f %c", name, &age, &rate, &c);`

- `printf("%s %d %f %c", name, age, rate, c);`
- `sprintf(table[i], "%s %d %f %c", name, age, rate, c);`

4

4

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)

Last lecture

- 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 UNIVERSITÉ UNIVERSITY

5

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

mnemonic: "expression *ptr is an int"


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


```
*ptr;  /* dereferencing. Indirect access. Alias of x.
          Get value of the pointee x  */
```

```
ptr    &x   address of x
*ptr    x   content (value) of x

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

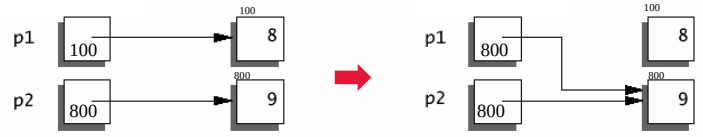
6

6

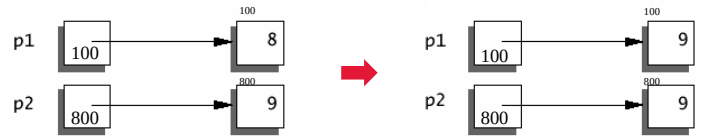
Some example of Pointers (summary)

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

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




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



7

7

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


(*p).data = x;
 () needed
```

8

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)

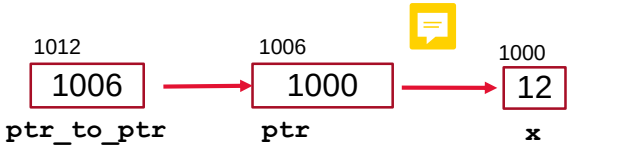
Last lecture

- 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 UNIVERSITÉ UNIVERSITY

Pointer to pointers

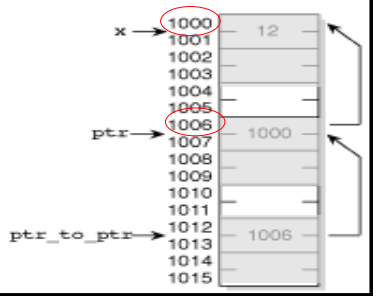
```
int x = 12;
int *ptr;
ptr = &x;
int **ptr_to_ptr;        /* I am a pointer to pointer */
ptr_to_ptr = &ptr;       /* points to ptr */
**ptr_to_ptr = 20;       /* multiple indirection*/
```



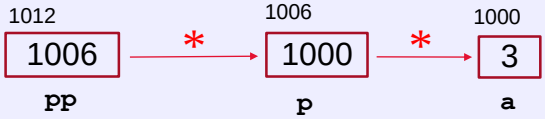
valid operations

```
x, &x                    *x
ptr   &ptr   *ptr        **ptr
ptr_to_ptr    &ptr_to_ptr
*ptr_to_ptr   **prt_to_ptr
```

```
**ptr_to_ptr == *ptr == x;
```



Multiple indirection



Consider the following code:

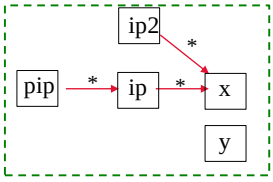
```
int a = 3;
int *p = &a;
int **pp = &p;      int **pp = &&a;
```

Here are how the values of these pointers equate to each other:

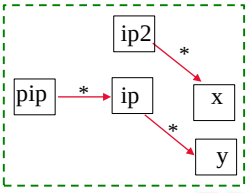
```
*pp == p == &a == 1000;
**pp == *p == a == 3;
```

11

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 ip

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




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

ip2 = pip; ???  Not valid!
pip = ip2; ???  Not valid!
```

12

12

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)

Last lecture

- 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 UNIVERSITÉ UNIVERSITY

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 argument? 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. Input / output

YORK UNIVERSITÉ UNIVERSITY

Mr. Main

Hi, SWAP function, I have some manuscripts, stored in lockers (memory), and I want you to repaint them so their color swapped.

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 swapped?

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!**

15

An example. Not working.

```
void increment(int x, int y)
{

  x ++;
  y += 10;
}

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

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

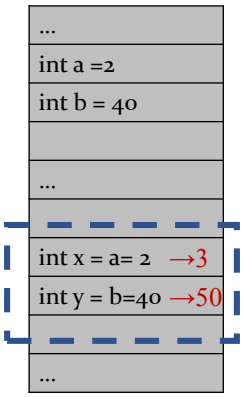
Pass by value !!!

x = a

y = b

2 40

running `main()`



16

The Correct Version

I am expecting int pointers

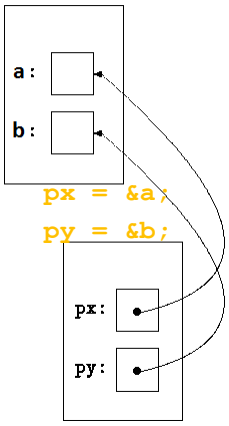
```
void increment(int *px, int *py)
{
   (*px) ++;  // *px is a

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

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

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

px = &a

py = &b

Pass by value !!!

in caller:



3 50

17

The Correct Version

I am expecting int pointers

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

   *py += 10;
}

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

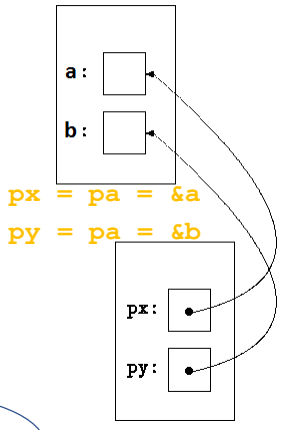
px = pa = &a

py = pa = &b

Pass by value !!!

Pass address/pointer, another way

in caller:



3 50

18

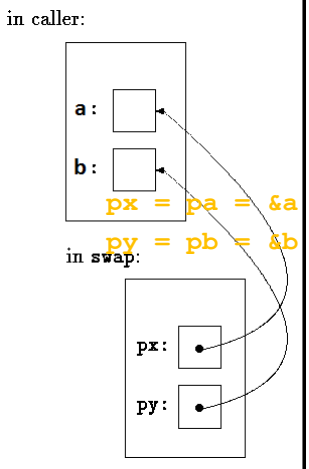
The Correct Version

I am expecting int pointers

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

void main( ) {
    int a=2, b=40;
    int *pa = &a;
    int *pb = &b;
    swap(pa,pb); // or swap(&a,&b);

}
```

px = pa = &a;
py = pb = &b

Pass by value !!!



19

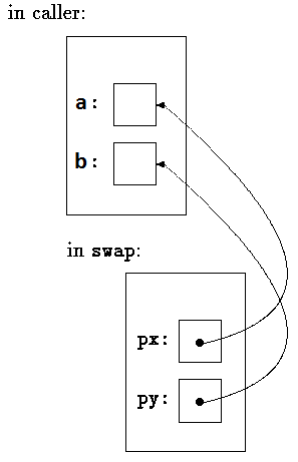
40 2

19

Another example

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

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

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

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

increment(px, py);



20

41 12

20

Now understand scanf() -- more or less

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

int x; char c;
scanf ("%d %c", &x, &c);
printf("%d %c", x, c);

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

But why array name is used directly

```
scanf ("%s %d", name, &age);
fgets (input, 5,stdin);
sscanf(input,"%s %d",name,&age);
```

explain shortly

21

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)

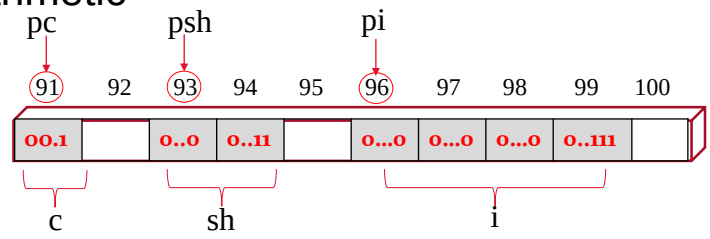
Last lecture

- 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 UNIVERSITÉ UNIVERSITY

22

Pointer arithmetic



- Limited math on a pointer
- Four arithmetic operators that can be applied

  +n -n ++ --

  Result is a new pointer (address)

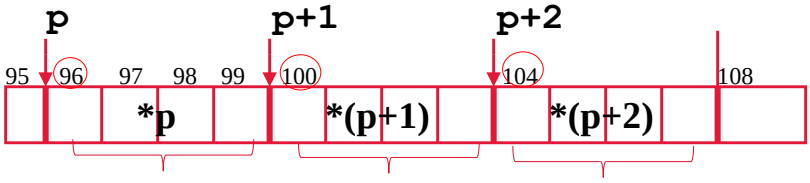
```
int* pi=&i;//96 char* pc=&c;//91   short* psh=&sh;//93

pi + 1?    97?
psh + 2?   95?
pi++?  pc++?  psh++?
```


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 type
  - If p points to an integer (4 bytes), value of unit is 4

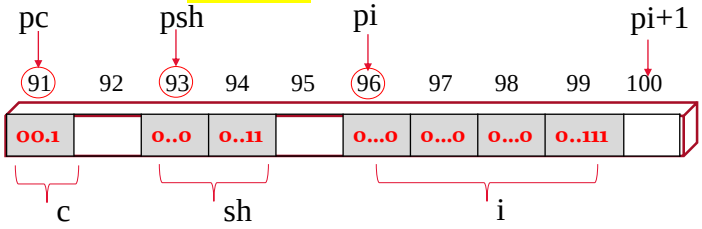
    p + n advances by n*4 bytes:

    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"*

Pointer arithmetic -- scaled




```
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 +2? address 93+ 2*2 =97

pi ++? **pc**++? **psh** +=3?

25

pi = 96 + 4 **pc = 91 + 1** **psh = 93 + 3*2**

YORK UNIVERSITÉ UNIVERSITY

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)

Last lecture

YORK UNIVERSITÉ UNIVERSITY

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) + - ++ --

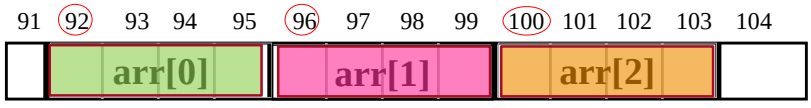
- **Pointers and arrays (5.3)**
  - Arrays are stored consecutively
  - Pointer to array elements p + i = &a[i] *(p+i) = a[i]
  - Array name contains address of 1st element a = &a[0]
  - Pointer arithmetic on array (extension)
  - Array as function argument – "decay"
  - Pass sub_array

YORK UNIVERSITÉ UNIVERSITY

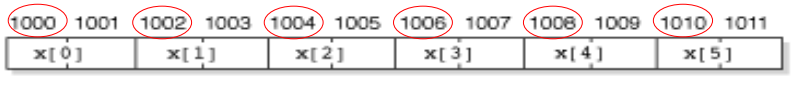
27

Pointers and Arrays (5.3)

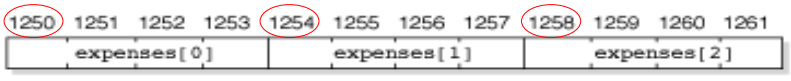
- Array members are next to each other in memory
  - `arr[0]` always occupies in the lowest address
  - `&arr[i+1] == &arr[i] +` size of type in byte;



short x[6];



float expenses[3];



28

Pointers K&R Ch 5

- Basics: Declaration and assignment
- Pointer to Pointer
- Pointer and functions (pass pointer by value)
- Pointer arithmetic +- ++ --

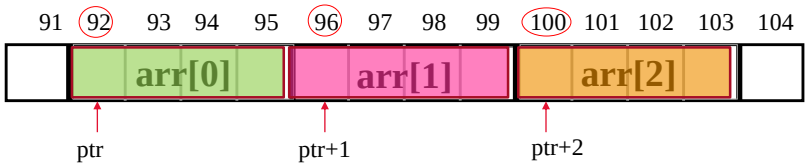
- **Pointers and arrays (5.3)**
  - Stored consecutively
  - Pointer to array elements p + i = &a[i] *(p+i) = a[i]
  - Array name contains address of 1st element a = &a[0]
  - Pointer arithmetic on array (extension)
  - Array as function argument – "decay"
  - Pass sub_array

YORK UNIVERSITÉ UNIVERSITY

29

Pointers and Arrays (5.3)

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




```
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]
```

YORK UNIVERSITÉ UNIVERSITY

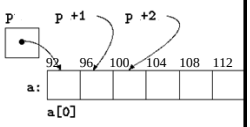
30

30

Sum of an int array

```
#define N 10

int arr[N], sum, i;
sum = 0;

for (i=0; i<N; i++)
    sum += arr[i];
```

```
#define N 10
int arr[N],sum, i,*p;
p = &arr[0]; // 92

sum = 0;
for (i=0; i< N; i++)
    sum += *(p + i);
```




```
ptr = &arr[0];
```

⇩

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

e.g. `*ptr == arr[0]`

```
#define N 10
int arr[N], sum, i,*p;
p = &arr[0]; // 92

sum = 0;
for (i=0; i< N; i++){
    sum += *p;        //*p++

    p++; // advance p (by 4 bytes)

}         // 92 96 100 ..
```

31

Pointers K&R Ch 5

- Basics: Declaration and assignment
- Pointer to Pointer
- Pointer and functions (pass pointer by value)
- Pointer arithmetic +- ++ --

- **Pointers and arrays (5.3)**
  - Stored consecutively
  - 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 UNIVERSITÉ UNIVERSITY

32

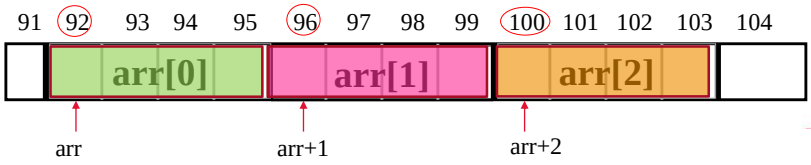
Pointers and Arrays (5.3)

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

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

- Identifier (name) of an array is equivalent to the address of its 1st element. `arr == &arr[0]`

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



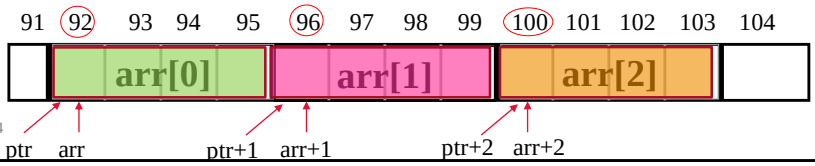
33 Array name can be used as a pointer. Follow pointer arithmetic rule!

Pointers and Arrays (5.3)

- 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]
```

```
int arr[3];
int * ptr;
ptr = arr;    /* == (ptr = &arr[0]) */
ptr + i == &arr[i]
*(ptr + i) == arr[i]
```



34

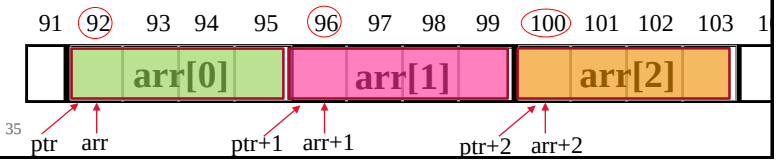
```
int arr[3]; int * p;
ptr = arr;    /*    ptr = &arr[0]   */
```

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

```
arr[i]
*(ptr + i)
*(arr + i)
ptr[i];
```

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

equivalent



35

Sum of an int array

```
#define N 10

int arr[N], sum, i;
sum = 0;

for (i=0; i< N; i++)
  sum += arr[i];
  sum += *(arr+i);
```

Compiler

```
#define N 10
int arr[N],sum, i,*p;
p = arr; // p=&arr[0] 92

sum = 0;
for (i=0; i< N; i++)
    sum += *(p + i);
```

```
ptr = arr;
```

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

e.g.,*ptr == *arr = a[0]

36

```
#define N 10
int a[N], sum, i,*p;
p = arr    /* p=&arr[0] */

sum = 0;
for (i=0; i< N; i++) {
    sum += *p;

    p++; // advance p (by 4 bytes)
}
```

36

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*/
```

37

37

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

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

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

```
strlen(arr);  /*valid*/     sizeof arr ?   10
strlen(p);    /*valid*/     sizeof p ?     8
```

38

Later today | same | Not same! | Stopped here last time

38

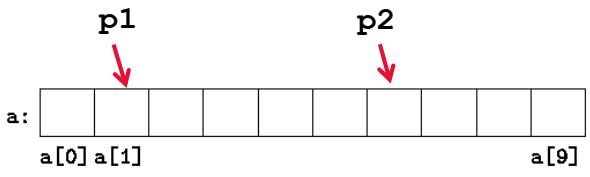
Pointers K&R Ch 5

- Basics: Declaration and assignment
- Pointer to Pointer
- Pointer and functions (pass pointer by value)
- Pointer arithmetic + - ++ --

- **Pointers and arrays (5.3)**
  - Stored consecutively
  - 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

Pointer arithmetic (revisit + extension)

- `+n -n ++ --`
- If `p1, p2` points to different elements of the same array
  - Differencing: `p1 – p2`
    result is how far apart in term of # elements
  - Comparison : `== != > < >= <=`
    `p1 < p2` is true (1) if `p1` points to earlier elements than `p2`



40

Pointer arithmetic on arrays (revisit)
Adding an Integer to a Pointer `+i`

- Adding an integer `i` to a pointer `p` yields a pointer to the element `i` places after the one that `p` points to.

- More precisely, if `p` points to the array element `a[k]`, then `p + i` points to `a[k+i]`.
  - `If p = &a[k]`
  - `Then p + i == &a[k+i]`

  Special case k=0:
  - `If p=a`
  - `Then p+i==&a[i]`

Pointer arithmetic on arrays (revisit)
Subtracting an Integer to a Pointer `-i`

- Subtracting an integer `i` to a pointer `p` yields a pointer to the element `i` places before the one that `p` points to.

- More precisely, if `p` points to the array element `a[k]`, then `p - i` points to `a[k-i]`.
  - `If p = &a[k]`
  - `Then p - i == &a[k-i]`

- Assume that the following declarations are in effect:

```
int a[10], *p, *q, i;
```

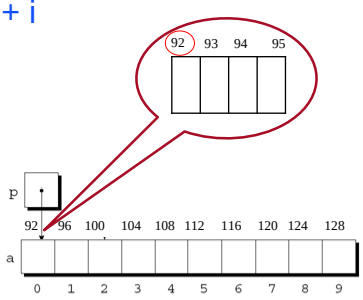
Pointer arithmetic on arrays (revisit)

Adding an Integer to a Pointer p + i

if p points to the array element a[k],
then p + i points to a[k+i].

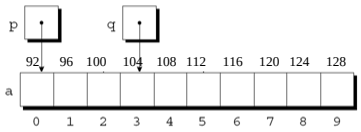
- Example of pointer addition:

```
p = a; // &a[0]; k=0
```



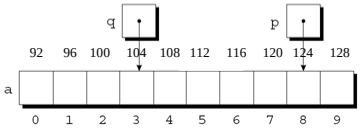

```
q = p + 3; // q points to a[3]
```

p= &a[0] = 92
Then q = 92+ 3×4 =104 = &a[3]




```
p += 8; // p points to a[8]
```

p= &a[0] = 92
Then p = 92+ 8×4 =124 = &a[8]



43

43

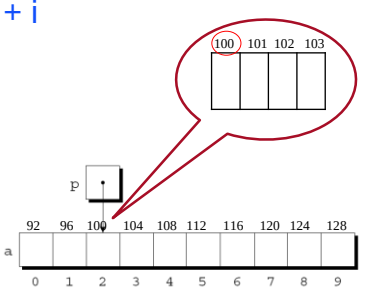
Pointer arithmetic on arrays (revisit)

Adding an Integer to a Pointer p + i

if p points to the array element a[k],
then p + i points to a[k+i].

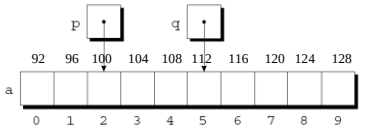
- Example of pointer addition:

```
p = &a[2]; // k=2
```



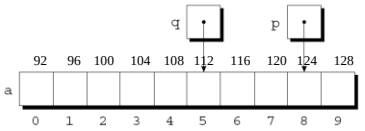

```
q = p + 3; //q points to a[2+3]
```

p= &a[2] = 100
Then q = 100+ 3×4 =112 = &a[5]




```
p += 6; // p points to a[2+6]
```

p= &a[2] = 100
Then p = 100+ 6×4 =124 = &a[8]



44

44

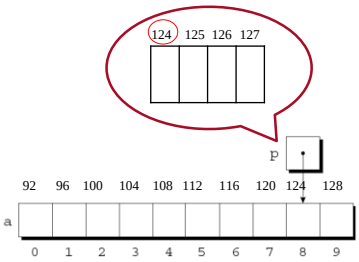
Pointer arithmetic on arrays (revisit)

Subtracting an Integer to a Pointer `p - i`

if `p` points to the array element `a[k]`,
then `p - i` points to `a[k-i]`.

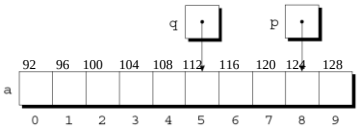
- Example:

```
p = &a[8]; // k=8
```



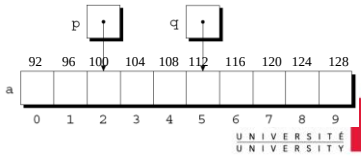

```
q = p - 3; // q points to a[8-3]
```

p= &a[8] = 124
Then q = 124 - 3×4 =112 = &a[5]




```
p -= 6; // p points to a[8-6]
```

p= &a[8] = 124
Then p = 124 - 6×4 =100 = &a[2]



45

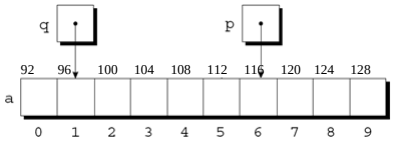
45

Pointer arithmetic on arrays (extended)

Subtracting One Pointer from another $p_1 - p_2$

- When one pointer is subtracted from another, the result is the distance (measured in array elements) between the pointers.
- If `p` points to `a[i]` and `q` points to `a[j]`, then `p - q` is an integer, equal to `i - j`.

```
p = &a[6];   // 116
q = &a[1];   // 96
```




```
int d = p - q;   // 5:   (116-96)/4 = 5 == 6-1
int d = q - p;   // -5:  (96-116)/4 = -5
```

Why designed this way?

46

46

Pointer arithmetic on arrays (extended) Comparing Pointers

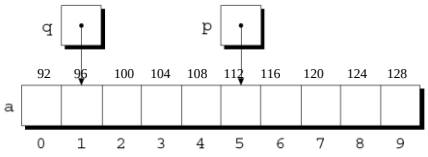
- Pointers can be compared using the relational operators **(< <= > >=)** and the equality operators (== and !=).
  - Using relational operators is meaningful only for pointers to elements of the same array.

- The outcome of the comparison depends on the relative positions of the two elements in the array.

```
p = &a[5];
q = &a[1];
```

`p <= q` is 0 "false"

`p >= q` is 1. "true"



47

47

Summary of pointer arithmetic

- Legal:
  - assignment of pointers of the same type p2 = p1
  - adding or subtracting a pointer with an integer p++ , p+2, p-2
  - subtracting or comparing two pointers to members of the same array p2- p1, if (p1 < p2) while (p1 != p2)
  - assigning or comparing to zero (NULL) p = NULL p==NULL

- Illegal: p1+p2; p1*p2; p1*3
  - add two pointers, multiply or divide two pointers, integers
  - shift or mask pointer variables p1 <<2 p1 | 3
  - add float or double to pointers p1 + 1.23
  - assign a pointer of one type to a pointer of another type (except for void *) without a cast

48

Sum of an int array

```
#define N 10

int arr[N], sum, i;
sum = 0;

for (i=0; i<N; i++)
  sum += arr[i];
  sum += *(arr+i);
```
compiler

```
#define N 10
int arr[N],sum, i,*p;
p = arr; // p=&arr[0]

sum = 0;
for (i=0; i< N; i++)
    sum += *(p + i);
```

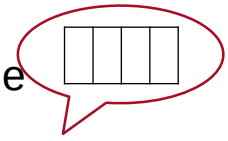
```
#define N 10
int arr[N], sum, i,*p;
p = arr;

sum = 0;
for( i=0;i< N; i++) {
    sum += *p;

    p++; // advance by 4

}
```

```
#define N 10
int arr[N], sum, *p;
p = arr;

sum = 0;
while( p <= arr+N-1 ) {
    sum += *p;

    p++;

}  // no i needed
```

&arr[N-1]
Address of last element

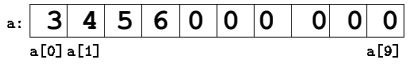
49

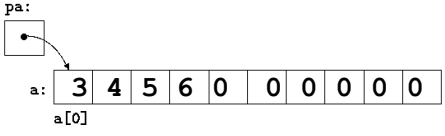
Pointers and Arrays: an Example

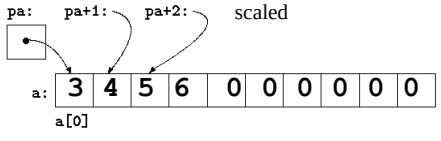


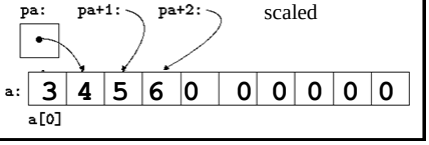
```
int a[10]={3,4,5,6};
int *pa;
pa = a; // pa=&a[0]

int x,y,z;

x = *pa;     // x = *a
     //same as x = a[0]

y = *(pa + 1);// pa+4 a[1]
z = *(a + 2); // a[2]

pa++;  // pa=&a[1]
x = *(pa+2)    ?
*(pa + 3) = 200;
```

x:6 y:4 z:5
a:3 4 5 6 200 0 0 .. 0









50

Pointers K&R Ch 5

- Basics: Declaration and assignment
- Pointer to Pointer
- Pointer and functions (pass pointer by value)
- Pointer arithmetic +- ++ --
- Pointers and arrays (5.3)
  - Stored consecutively
  - 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) p1-p2 p1<>!= p2
  - Array as function argument – "decay"
  - Pass sub_array
- Array of pointers
- Command line argument
- Pointer to arrays and two dimensional arrays
- Pointer to functions
- Pointer to structures
- Memory allocation file IO

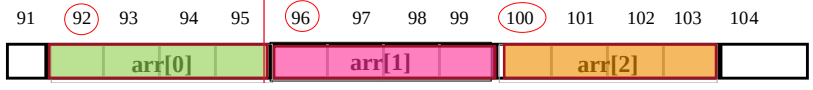
Last lecture + So far

YORK UNIVERSITÉ UNIVERSITY

51

Summary

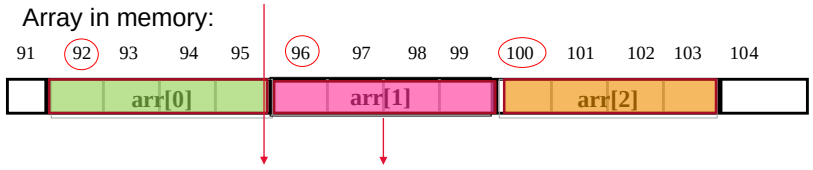
- Pointer arithmetic: If p points to an integer of 4 bytes, p + n advances by 4*n bytes: p + 1 = 96 + 1*4 = 100 p + 2 = 96 + 2*4 = 104
- Array in memory:



52

Summary

- 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
- Array in memory:

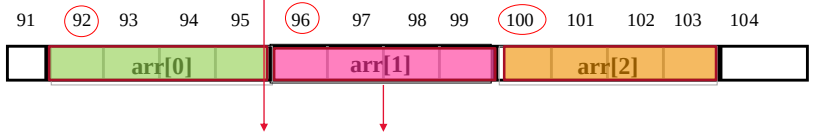


- Suppose `p` points to array element `k`, then `p+1` points to `k+1` (next) element.
  `p + i` points to `arr[k+i]`.
  - `p = &arr[k]:  p + i == &arr[k+i]  → *(p+i) == arr[k+i]`
  - `k=0: p=&arr[0]:  p + i == &arr[i]  → *(p+i) == arr[i]`

53

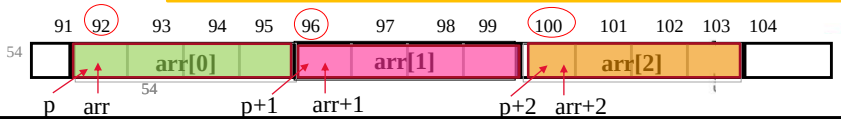
Summary

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

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



54

Summary

arr can be used as a pointer

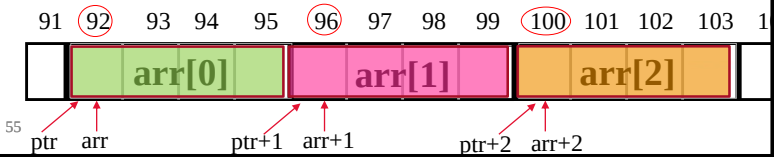
```
int arr[3]; int * p;
ptr = arr;    /*    ptr = &arr[0]   */
```

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

```
arr[i]
*(ptr + i)
*(arr + i)
ptr[i];
```

equivalent

Compiler convert arr[2] to *(arr+2)



55

- Some interesting facts so far
  - p + n is scaled "for int * p, p+1 is p+4"
  - p1 – p2 is scaled `(116-96)/4 = 5`
  - Array name contains address of its first element a == &a[0]
- Why designed this way?
  - Facilitate Passing Array to functions!
  - We will see how.
- we will also look into, under call-by-value,
  - how array can be passed to function
  - how does `strcpy(arr, arr2), strcat(arr,arr2)` etc modify argument array

YORK UNIVERSITÉ UNIVERSITY

56

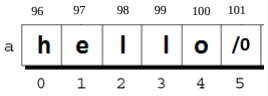
Pointers K&R Ch 5

- Basics: Declaration and assignment
- Pointer to Pointer
- Pointer and functions (pass pointer by value)
- Pointer arithmetic +- ++ --
- Pointers and arrays (5.3)
  - Stored consecutively
  - 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) p1-p2 p1<>!= p2
  - **Array as function argument – "decay"**
  - Pass sub_array
- Arrays of pointers
- Command line argument
- Pointer to arrays and two dimensional arrays
- Pointer to functions
- Pointer to structures
- Memory allocation

Last lecture

New today

Arrays Passed to a Function



- The name/identifier of the array passed is actually a pointer/address to its first element. arr == &arr[0];

```
char a[20] = "Hello";
strlen(a); /* strlen(&a[0]). 96 is passed */
```

- The call to this function does not copy the whole array itself, just a *address* *(starting address -- a single value)* to it.

- Thus, function expecting a char array can be declared as either

```
strlen(char s[]);
   or
strlen(char * s);
```

Actual prototype man 3 strlen

String library functions

- Defined in standard library, prototype `<string.h>`

- `unsigned int strlen(char *)`
  - `# of chars before first '\0'`
  - `not counting '\0'`

- `strcpy (char * toStr, char * fromStr)`
  - `strncpy(toStr,fromStr,n)`
  - `modify toStr`

- `strcat(char * s1, char * s2)`
  - `strncat (s1, s2, n)`
  - `modify s1`

- `int strcmp(char * s1, char * s2)`
  - `strncmp(s1,s2,n)`

59

59

Other String process library functions

- Defined in standard library, prototype `<stdlib.h>`

- `int atoi(char *)`

- `long atol(char *)`

- `double atof(char *)`

```
char arr[] = "134";
int a = atoi(arr)
```

60

60

Function processing general arrays

Description
The C library function **qsort** sorts an array.

Declaration
void **qsort** (void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))

Parameters
- **base** – This is the pointer to the first element of the array to be sorted.
- **nitems** – This is the number of elements in the array pointed by base.
- **size** – This is the size in bytes of each element in the array.
- **compar** – This is the function that compares two elements.

Description
The C library function **bsearch** searches an array of **nitems** objects

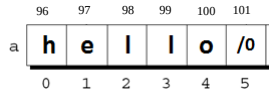
Declaration
void * **bsearch** (const void *key, const void *base, size_t nitems, size_t size, int (*compar)(const v
const void *))

Parameters
- **key** – This is the pointer to the object that serves as key for the search, type-casted as a void*
- **base** – This is the pointer to the first object of the array where the search is performed, type-casted as a void*.
- **nitems** – This is the number of elements in the array pointed by base.
- **size** – This is the size in bytes of each element in the array.
- **compar** – This is the function that compares two elements.

For your information

61

Arrays Passed to a Function



- Thus, function expecting a char array can be declared as either

```
strlen(char s[]);
```

or

```
strlen(char * s);
```

Actual prototype man 3 strlen

- The call to this function does not copy the whole array itself, just a *address* *(starting address -- a single value)* to it.

```
char a[20] = "Hello";
char * ps = a;
strlen(a); /* strlen(&a[0]). 96 is passed */
strlen(ps);
```

"decay"

Pass by value: 96 is passed and copied to s

```
s = a = &a[0] //s is a local pointer variable
s = ps = a = &a[0] // in function
```

62

62

Arrays Passed to a Function

- Thus, function expecting a char array can be declared as either

```
strcpy(char dest[], char src[]);
```

or

```
strcpy(char * dest, char * src);
```

Actual prototype man 3 strcpy

- The call to this function does not copy the whole array itself, just a *address (starting address -- a single value)* to it.

```
char a[20];  char b[20] = "hi";
char * ptrA = a; char * ptrB = b;
```

"decay"

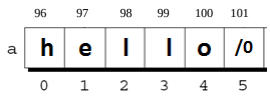
```
strcpy(a, b)   /* strcpy(&a[0], &b[0])*/
strcpy(ptrA, ptrB);
```

```
scanf("%s", a);        printf("%s",a);
scanf("%s", ptrA);     printf("%s", ptrA);
```

63

63

Arrays Passed to a Function



- Arrays passed to a function are passed by starting address.

- The name/identifier of the array passed is treated as a pointer to its first element. arr = &arr[0];

"decay"

By passing an array by a pointer (its starting address)

1. Array can be passed (efficiently)
   - a single value (e.g, 96, no matter how long array is)
2. Argument array can be modified
   - no & needed

```
strcpy(arr, "hello");
scanf("%s %d %f %c", arr, &age, &rate, &c);
sscanf (table[i], "%s %d %f %c", name,&age,&rate,&c)
```

64

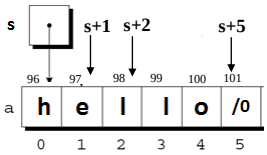
64

Examples using prior knowledge

Computing String Lengths -- Access argument array

```
int strlen(char *s) //s = arr == &arr[0] 96 passed by value
{                   // access arr
   int n=0;
   while ( *(s+n) != '\0')
   {
     n++;
   }
   return n;
}
```

compiler




```
int strlen(char s[])
{
  int n=0;
  while (s[n] != '\0')
  {
    n++;
  }
  return n;
}
```

```
char * ptr = arr;
strlen(arr);   /* s==arr==&arr[0]. arr 'decayed' to 96 */
strlen(ptr);   /* s== ptr == arr == &arr[0] */
```

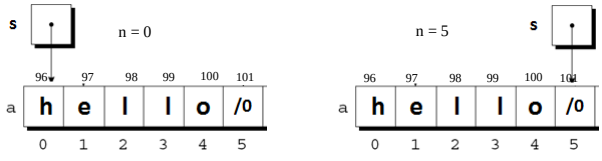
Function receives a single address value.
Does not know/care if it an array or not

Examples using prior knowledge

Computing String Lengths -- another version

```
/* move the pointer s */
int strlen(char *s)    /* s = arr == &arr[0] 96 passed */
{
   int n =0;
   while (*s != '\0'){
     n++;
     s++;    // move s (by 1),jumping to next element of arr
   }
   return n;
}
```

Function receives a single address value.
Does not know/care if it an array or not

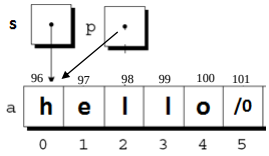



```
char * p = arr;
strlen(arr);  /*  s==arr==&arr[0] */
strlen(ptr);  /*  s== prt == arr == &arr[0] */
```

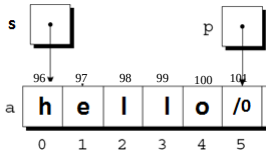
Examples using prior knowledge

Computing String Lengths -- 'cool' version A

```
/* strlen: return length of string s */
int strlen(char *s)
{
    char *p = s;
    while ( *p != '\0')
       p++;
    return p - s; // how far apart? (101-96)/1=5
}
```



Don't need n, n++, potentially faster



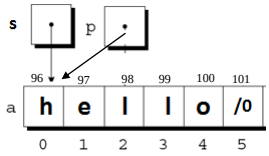

```
char * p = arr;
strlen(arr);
strlen(ptr);
```

67

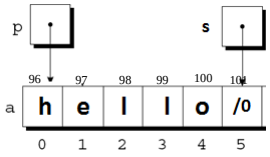
Examples using prior knowledge

Computing String Lengths -- 'cool' version B

```
/* strlen: return length of string s */
int strlen(char *s)
{
    char *p = s;
    while ( *s != '\0')
       s++;
    return s - p; // or abs(p-s)  how far apart? (101-96)/1=5
}
```



Don't need n, n++, potentially faster

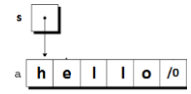



```
char * p = arr;
strlen(arr);
strlen(ptr);
```

68

Examples using prior knowledge

Modify argument arrays




```
int processArr(char *s) // s=arr == &arr[0] call by value
{  int i=0;
   int len = strlen(s)
   while ( i < len)
   {
     *(s+i)='X';
     i++;
   }}
```

compiler

```
void processArr(char s[])
{   int i=0;
    int len=strlen(s);
    while ( i< len)
    { s[i] = 'X';
      i++;
    }        // s[i++]='X';
 }
```

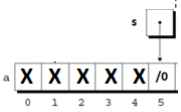
```
int processArr(char *s)
{
   int i=0;
   while ( *(s+i) != '\0')
   {   *(s+i)='X';
       i++;
   }
```

No strlen()

Traverse just once

```
int processArr(char *s)
{
    while ( *s != '\0' ){
       *s = 'X';
       s++;
   }
}
```

Move s, no i



copy strings – access one, modify one

```
/* strcpy: copy two strings */
void strcpy(char *dest, char *src) /* or (char s[]) */
{
   while (1){
     *dest = *src;
     if (*dest = '\0')
       return;
     src ++;
     dest ++;
   }
}
```

```
void stringcopy(char dest [], char src [])
{
   int i=0;
   while (1){
      dest[i] = src[i];
      if (src[i] == '\0')
           break;

      i++;
   }
}
```

Compiler:

(dest+i)=(src+i)

\0 is also copied

Another way writing

```
void strcpy(char *dest, char *src)
{
   while ( (*src = *dest) != '\0')
   {   src++; dest++;    }
}
```

Pointers K&R Ch 5

- Basics: Declaration and assignment
- Pointer to Pointer
- Pointer and functions (pass pointer by value)
- Pointer arithmetic +- ++ --

- **Pointers and arrays (5.3)**
  - Stored consecutively
  - 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

Passing Sub-arrays to Functions

- It is possible to pass part of an array to a function, by passing a pointer to the beginning of the sub-array.

```
my_func( int arr[ ] )  {...}
or
my_func( int *arr )  {...}
```

```
caller
my_func(&a[5])
or
my_func(a + 5)
```


Passing Sub-arrays to Functions

- It is possible to pass part of an array to a function, by passing a pointer to the beginning of the sub-array.

```
char arr[20] = "hi world";
char * p = arr;  // &arr[0]
strlen(p);
strlen(arr);
```

Functions receive address 92

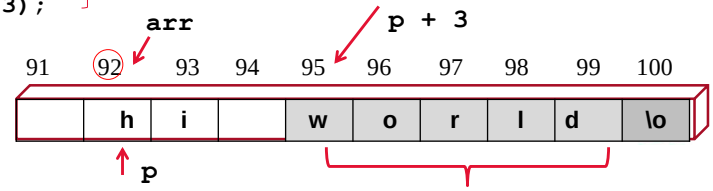
8

```
strlen (&arr[3]);
strlen (arr + 3);
strlen (p + 3);
```

Functions receive address 95

5

&arr[3]
arr + 3



73

```
printf("%s",pa + 3);?   // world
```

sub-array

73

Passing Subarrays to Functions -- Recursion

```
int length (String s) // Java
   if ( s.equals("") contains no letter)
     return 0;
   return 1 + length(s.substring(1));
}
```

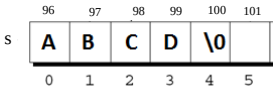
```
length("ABCD")
= 1 + length("BCD")
= 1 +  ( 1 + length("CD"))
= 1 +  ( 1 +  ( 1 + length("D")))
= 1 +  ( 1 +  ( 1 +  (1+ (1+length("") )))
= 1 +  ( 1 +  ( 1 +  (1+ (1+0) ))) = 4
```

C version?

74

74

Passing Subarrays to Functions -- Recursion




```
length("ABCD")
= 1 + length("BCD")
= 1 + ( 1 + length("CD"))
= 1 + ( 1 + ( 1 + length("D")))
= 1 + ( 1 + ( 1 + (1+ (1+length(""))))
= 1 + ( 1 + ( 1 + (1+ (1+0) ))) = 4
```

```
int main(){
    char s[] = "ABCD";
    int len = length(s);
    printf("%d",len); // 4
}

int length(char * c){
    if (*c == '\0')
      return 0;
    else
      return 1 + length(c + 1);
}
```

97 98 99 100

75

Array Arguments (Summary)

"decay"

- The fact that an array argument is passed by a pointer (its starting address) has some important consequences.

- Consequence 1:
  - Due to 'pass by value', when an ordinary variable is passed to a function, its value is copied; any changes to the corresponding parameter don't affect the variable.

  - In contrast, by passing array by pointer, argument array can be modified

```
void processArr(chars[]) // no &
strcpy (message, "hello");   // no &
scanf ("%s", message);  // no &
```

76

YORK UNIVERSITÉ UNIVERSITY

76

Pointers and arrays (Summary)

- Consequence 2:
  - The time required to pass an array to a function doesn't depend on the size of the array. There's no penalty for passing a large array, since no copy of the array is made.
- Consequence 3:
  - An array parameter can be declared as a pointer if desired.

```
strlen (char * s)
processArr(int *s)
```

- Consequence 4:
  - A function with an array parameter can be passed an array "slice" — substring

```
strlen (&a[6]),
strlen (a + 6)
```

77

77

Array Arguments (Summary)

"decay"

- An array argument is passed by a pointer (its starting address). Thus the called function just receives an single address, has no view of the whole array
  - efficient

- Then how does the function know where to stop?
- For char [], rely on `'\0'`, but how about general arrays?

```
// find the max value in the array
int findMax (int c[]){ // (int * c)

}
```

78

78

General array as function argument

- Pass an array / string by only the address / pointer of the first element
  - `strlen("Hello");`
- You need to take care of where the array ends, the function does not know if it is an array or just a pointer to char/int

"decay"

- Two possible approaches:
  1. Special token/sentinel/terminator at the end (case of "string" `'\0'`)
  2. Pass the length as another parameter

Function: `arrayLen(int *)` `arraySum(int *)` `findMax(int *)`

Caller: `int a[20]; arrLen(a); arraySum(a); findMax(a);`

79

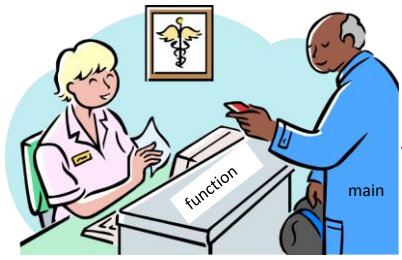
79

```
int main(){
   int arr [] = {1,2,3,4,5,6};
   int siz = sizeof(arr); // 6*4=24
   int len = sizeof(arr)/sizeof(int); // 24/4 = 6:

   a = findMax(arr);
   …
}

/* find max in the int array */
int findMax (int c[]){ // (int * c)

   int len = sizeof(c)/sizeof(int);  //  8/4=2
   int max = c[0]; i=1;
   while ( i <  len ){
      if (c[i]> max)
        max = c[i]
      i++;
   }
   return max;
```

80

80

strlen(char *) **arrayLen(int *, int n)**



Mr. Main:
Hi, Mrs binding function, I have some manuscripts, stored in lockers (memory), and I need you to bind them into a book. Could I bring the manuscripts to you?

Ms function:
Hi, Mr Main, here is how we work:
First, we don't take your original manuscript (not pass by reference). We always photocopy things (call by value), and work on copies.
Second, we only photocopy one paper a time(a single value)

Mr. Main:
Then, is there a way to have my original papers bound by you?

Ms function:
Write down the locker number (starting 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 bind them!

Ms function:
Well, then you also need to tell us where to stop fetching.
Either 1) tell us how many lockers to fetch, or, 2) put a special token in the last locker

Mr. Main:
My manuscripts are in multiple (consecutive) lockers

81

```
int main(){
   int arr [] = {1,2,3,4,5,6};
   int siz = sizeof(arr); // 6*4=24
   int len = sizeof(arr)/sizeof(int); // 24/4 = 6;

   a = findMax(arr, len);
   …
}

/* find max in the int array. */
int findMax (int c[], int leng){
   int max = c[0]; i=1;
   while ( i <  leng )
      if (c[i]> max)
        max = c[i]
      i++;
   return max;
}
```

82

YORK UNIVERSITÉ UNIVERSITY

82

```
int main(){
    int arr [] = {1,2,3,4,5,6};
    int siz = sizeof(arr); // 6*4=24
    int len = sizeof(arr)/sizeof(int); // 24/4 = 6;

    a = findMax(arr, len);
    …
}

/* Pointer version */
int findMax (int *c, int leng){
    int max = *c; // c[0]
    int i=1;
    while ( i <  leng )
        if (*(c+i)> max)
          max = *(c+i)
        i++;
    return max;
```

int c[] is converted to int * c by compiler

compiler

c[i] is converted to * (c+i) by compiler

83

83

```
int main(){
    int arr [] = {1,2,3,4,5,6};
    int siz = sizeof(arr); // 6*4=24
    int len = sizeof(arr)/sizeof(int); // 24/4 = 6;

    a = findMax(arr, len);
    …
}

/* Pointer version */
int findMax (int *c, int leng){
    int max = *c; // c[0]
    int i=1;
    while ( i <  leng )
        if (*c > max)
          max = *c;
        c++;  // c advances 4, pointing to next element
    return max;
```

Moving c

84

84

Function processing general arrays

Description
The C library function **qsort** sorts an array.

Declaration
void **qsort** (void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))

Parameters
- **base** – This is the pointer to the first element of the array to be sorted.
- **nitems** – This is the number of elements in the array pointed by base.
- **size** – This is the size in bytes of each element in the array.
- **compar** – This is the function that compares two elements.

Description
The C library function **bsearch** searches an array of **nitems** objects

Declaration
void * **bsearch** (const void *key, const void *base, size_t nitems, size_t size, int (*compar)(const v
const void *))

Parameters
- **key** – This is the pointer to the object that serves as key for the search, type-casted as a void*
- **base** – This is the pointer to the first object of the array where the search is performed, type-casted as a void*.
- **nitems** – This is the number of elements in the array pointed by base.
- **size** – This is the size in bytes of each element in the array.
- **compar** – This is the function that compares two elements.

For your information

85

Java avoids the hassle

```
public static void main(String[]args)
{
    int arr [] = {1,3,5,7,9,11};
    int a = findMax(arr);
    …
}

/* find max in the int array */
public static int findMax (int c[]){
    int max = c[0]; i=1;
    while ( i <  c.length ) {
        if (c[i] > max)
          max = c[i]
        i++;
    }
    return max;
```

Array object

arr	
value	1 3 5 7 9 11
length	6
……	

pass starting address (still call by value)

Starting locker has extra info: next 6 lockers

86

For your information

86

"Pointers" in Java

Not possible in Java

- No pointer accessible for primitive data types; swap(&a,&b)
- For arrays and objects, by "pointer" (reference) automatically!
  - Like pass array in C
- No dereference `* student`
- No address arithmetic

```
int strlen(char *s)
{  char * p = s;
   while ( *p != '\0')
      p++;
   return p – s;
}
```

Not possible in Java

- Safer, easier -- you don't need to worry about low level
- Slower (among other reasons)

For your information

Pointers K&R Ch 5

- Basics: Declaration and assignment
- Pointer to Pointer
- Pointer and functions (pass pointer by value)
- Pointer arithmetic +- ++ --
- Pointers and arrays (5.3)
  - Stored consecutively
  - 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) p1-p2 p1<>!= p2
  - Array as function argument – "decay"
  - Pass sub_array

- Arrays of pointers
- Command line argument
- Pointer to arrays and two dimensional arrays
- Pointer to functions
- Pointer to structures
- Memory allocation
- file IO

After midterm

YORK UNIVERSITÉ UNIVERSITY

- Written midterm next week in class
- "Midterm" page on course website soon
- What we have done so far?

What we have done so far

- Type, operators and expressions (Ch 2) :
  - Types and sizes
    - Basic types, their size and constant values (literals)
      - ✓ char: x >= 'a' && x <= 'z'; x >= '0' && x <= '9' avoid x>=48 && x<=57
      - ✓ int: 122, 0122, 0x12F convert between Decimal, Bin, Oct, Hex
    - Arrays (one dimension) and strings (Ch1.6,1.9)
      - ✓ "hello" has size 6 byte

H	e	l	l	o	\0

  - Expressions
    - Basic operators (arithmetic, relational and logical)
      - ✓ y=x++; y=++x;
      - ✓ !0 !-3 if (x = 2)
    - Type conversion and promotion 9/2*2.0 2.0*9/2 int i= 3.4
    - Other operators (bitwise, bit shifting , compound assignment, conditional)
      - ✓ Bit: |, &, ~. ^, << >>
      - ✓ Compound: x += 10; x >>= 10; x += y + 3
    - Precedence of operators flag | 1 << 3
- Functions and Program Structure (Ch 4)

What we have done so far
Ch 4

- C program structure, functions
  - Multiple files
  - Communication by global variables
  - “Call by value” increment() swap()
- Categories, scope and life time, initialization of variables (and functions)
  - global and local variables
  - static
- C Preprocessing
  - #include, #define
- Recursion

YORK UNIVERSITÉ UNIVERSITY

What we have done so far

K&R Ch 4, Appendix B

- Other C materials before pointer
  - Common library functions [Appendix of K&R]
  - 2D array, string manipulations
    - sscanf,sprintf,
    - fgets, fputs

YORK UNIVERSITÉ UNIVERSITY

What we have done so far

K&R Ch 5 Pointers

- Basics: Declaration and assignment
- Pointer to Pointer
- Pointer and functions (pass pointer by value)
- Pointer arithmetic +- ++ --
- Pointers and arrays (5.3)
  - Stored consecutively
  - 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) p1-p2 p1< > != p2
  - Array as function argument – “decay”
  - Pass sub_array

YORK UNIVERSITÉ UNIVERSITY