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

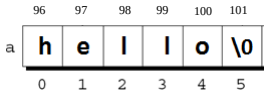
So far

- 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

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

Arrays passed to a Function



- The name/identifier of the array passed is actually a pointer/address to its first element. arr == &arr[0]; *fact3*

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

- The call to a 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)`

75

75

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)
```

76

76

String-related library functions

Basic I/O functions <stdio.h>

- **int printf (char *format, arg1, ….);**
  - Formats and prints arguments on standard output (screen or > outputFile)
  - **printf("This is a test %d \n", x)**
- **int scanf (char *format, arg1, ….);**
  - Formatted input from standard input (keyboard or < inputFile)
  - **scanf("%x %d", &x, &y)**

- **int sprintf (char * str, char *format, arg1,…..);**
  - Formats and prints arguments to str
  - **sprintf(str, "This is a test %d \n", x)**
- **int sscanf (char * str, char *format, arg1, ….);**
  - Formatted input from str
  - **sscanf(str, "%x %d", &x, &y)** // tokenize string str

YORK UNIVERSITÉ UNIVERSITY

77

Function processing general arrays <stdlib.h>

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

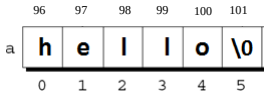
Declaration
void * **bsearch** (const void *key, const void *base, size_t nitems, size_t size, int (*compar)(const v
const void *))

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

78

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.

"decay"

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

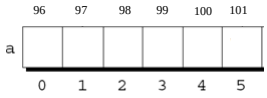
Pass by value: 96 is passed and copied to s

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

79

79

Arrays Passed to a Function

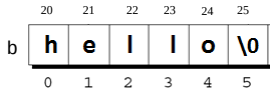


- Thus, function expecting a char array can be declared as either

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

or

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



Actual prototype

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

"decay"

```
char a[6];  char b[6] = "hello";
char * ptrA = a; char * ptrB = b;
strcpy(a, b)  /* strcpy(&a[0], &b[0]) */
```

dest = a = &a[0]
src = b = &b[0]

```
strcpy(ptrA, ptrB);
```

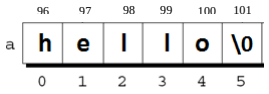
dest = ptrA = a = &a[0]
src = ptrB = b = &b[0]

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

80

80

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)
```

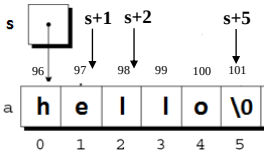
82

82

Examples using prior knowledge

Computing String Lengths -- Access argument array

```
int strlen(char *s) //s = arr == &arr[0] 96 passed by value
{                   // access arr
   sizeof(s)?

   int n=0;
   while ( *(s+n) != '\0')
   {
     n++;
   }
   return n;
}
```



compiler

```
int strlen(char s[])
{ sizeof(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

83

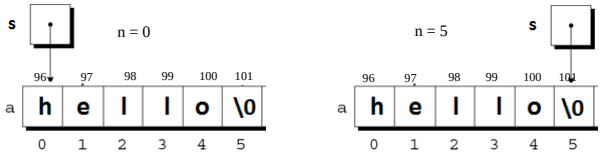
83

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] */
```

84

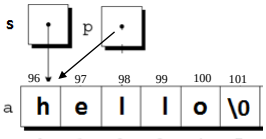
84

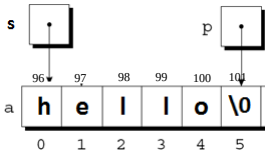
Examples using prior knowledge

Computing String Lengths -- 'cool' version A

```
/* strlen: return length of string s */
int strlen(char *s)
{
   char *p = s;
   while ( *p != '\0')
      p++;
}
```

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





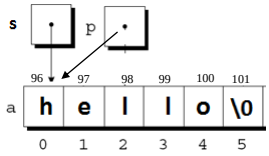

```
char * p = arr;
strlen(arr);
strlen(ptr);
```

85

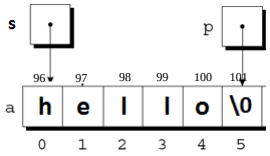
Examples using prior knowledge

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

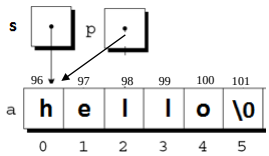



```
char * p = arr;
strlen(arr);
strlen(ptr);
```

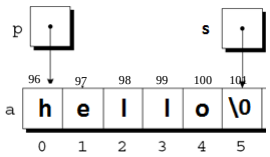

Examples using prior knowledge

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
}
```



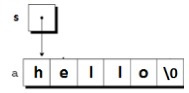
Don't need n, n++, potentially faster




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

Examples using prior knowledge

Modify argument arrays




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

Traverse twice

```
// s=arr == &arr[0] call by value
```

compiler

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

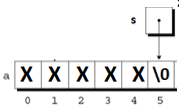
```
void processArr(char *s)
{
  int i=0;
  while ( *(s+i) != '\0')
  {   *(s+i)='X';
      i++;
  }
```

No strlen()

Traverse just once

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

Move s, no i



88

88

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++;    }
}
```

89

Array Arguments (Summary so far)

"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 &
```


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(char *s)
```

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

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**

today

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

YORK UNIVERSITÉ UNIVERSITY

92

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

94

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(&arr[0]);
strlen(arr);
strlen(p);     8
```

Functions receive address 92

```
printf("%s", p); // arr &arr[0]
```

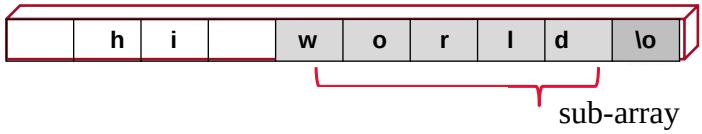
```
//length of world
strlen (        );
strlen (        );
strlen (        );
```

Functions receive address 95

```
&arr[3]
arr + 3
p + 3
```

5

```
print world?
```



sub-array

97

97

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(&arr[0]);
strlen(arr);
strlen(p);     8
```

Pointer/address level

Functions receive address 92

```
printf("%s", p); // arr &arr[0]
```

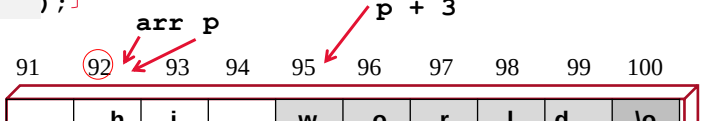
```
//length of world
strlen (        );
strlen (        );
strlen (        );
```

Functions receive address 95

```
&arr[3]
arr + 3
p + 3
```

5

```
print world?
printf("%s", p+3);?
printf("%s", p+5);?
```



sub-array

98

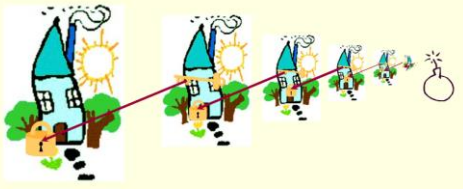
Passing Subarrays to Functions -- Recursion

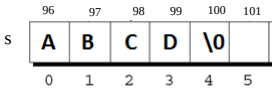

```
int length (String s) // Java
   if ( s.equals("") contains no letter)
     return 0;
   return 1 + length(s.substring(1));
}
```

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 

Passing Subarrays to Functions -- Recursion



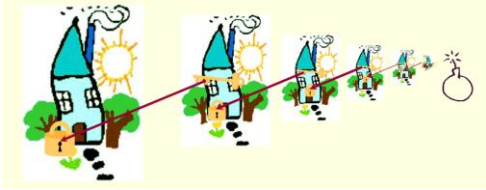


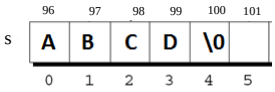
length("ABCD")
= 1 + length("BCD")
= 1 + (1 + length("CD"))
= 1 + (1 + (1 + length("D")))
= 1 + (1 + (1 + (1+ (1+length("")))))
= 1 + (1 + (1 + (1+ (1+0)))) = 4

```
int main(){
   char s[] = "ABCD";
   int len = length(s); //pass 96
   printf("%d",len); // 4
}

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

97 98 99 100

Passing Subarrays to Functions -- Recursion





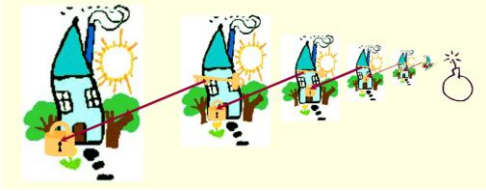

```
length("ABCD")
= 1 + length("BCD")
= 1 +  ( 1 + length("CD"))
= 1 +  ( 1 +   ( 1 + length("D")))
= 1 + ( 1 +   ( 1 +  (1+ (1+length(""))))
= 1 + ( 1 +   ( 1 +  (1+ (1+0) ))) = 4
```

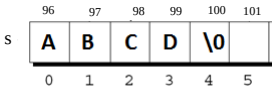
```
int main(){
   char s[] = "ABCD";
   int len = length(s); //pass 96
   printf("%d",len); // 4
}

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

97 98 99 100

102

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); //pass 96
   printf("%d",len); // 4
}

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

97 98 99 100

103

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 &
```


YORK UNIVERSITÉ UNIVERSITY

Pointers and arrays (Summary revisit)

- 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(char *s)
```

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

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

"Disadvantages"?

YORK UNIVERSITÉ UNIVERSITY

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 a char or int

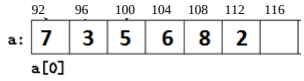
"decay"

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

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

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


```
int main(){
   int a [] = {7,3,5,6,8,2};

   int max = findMax(a);
   …
}
```




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

   int len = sizeof(arr)/sizeof(int);  //  8/4=2

   while ( i <  len ){
      …
```

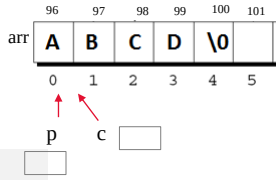
sizeof does not work in function

```
lab5E.c:66:28: warning: 'sizeof (arr)' will return the size of the pointer, not the array itself
      [-Wsizeof-pointer-div]
     int size = sizeof(arr)/sizeof(int);
```

Some nice compiler (MAC. not lab gcc ☹)

sizeof is not a function. It is an operator

```
int main(){
    char arr [] = "ABCD";
    char * p = arr;

    strlen(arr);
    strlen(p);

    sizeof arr;
    sizeof p;

    aFunction(arr);
    …
}

int aFunction (char c[]){ // (char * c)

    strlen(c);
    sizeof(c);
    …
```

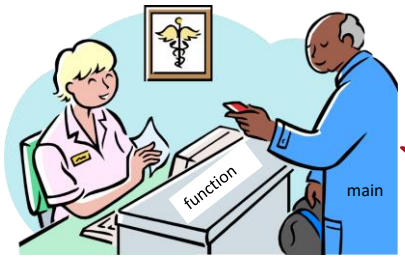


For length, sizeof does not work on pointer and in function

108

108

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?

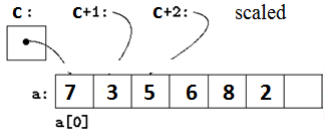
Ms function:
Write down the locker number (starting address) on a paper, bring that paper to us (pass pointer/address) we photocopy the paper (still pass by value),
Then, based on the locker number on the copy, we go to your locker, fetch your original manuscripts there and bind them!

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

109

```
int main(){
      int arr [] = {17,3,5,19,8,2};
      finaMax(arr, 6);
}

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



111

111

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.

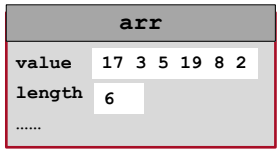
For your information

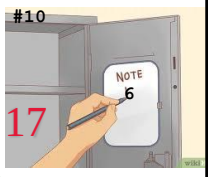
112

Java avoids the hassle


```
public static void main(String[]args)
{
    int arr [] = {17,3,5,19,8,2};
    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 ) {
        ......


    }
    return max;
```

Array object





Java also pass starting address (call-by-value)

For your information

YORK UNIVERSITÉ UNIVERSITY

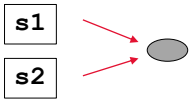
"Pointers" in Java

Not possible in Java

- No pointer accessible for primitive data types; swap(&a,&b)
- For arrays and objects, by "pointer" (reference) automatically!
  - `Student s1 = new Student();  Student s2 = s1;`
  - 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)

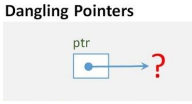
For your information

Problems with pointers

```
int *ptr;          /* I'm a pointer to an int */
ptr= &a            /* I got the address of a */
*ptr = 5;          /* set contents of the pointee a */
```

```
int *ptr;          /* I'm a pointer to an int */
*ptr = 5;          /* set contents of the pointee to 5 */
```

- `ptr` is uninitialized. "points to nothing". "dangling"
  Has some random value `0x7fff033798b0`
  - may be your OS!



- dereferencing an uninitialized pointer? Undefined behavior!

- Always make `ptr` point to sth! How?

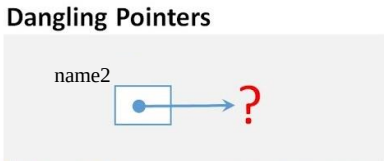
```
1) int a; ptr =&a;    int arr[20]; ptr=&arr[0];
2) ptr = ptr2   /* indirect. assuming ptr2 is
3) ptr = malloc (.....) /* later today? *
```

116

116

Problems with pointers, another scenario

```
char name[20];
char *name2;
int age; float rate;
```




```
printf("Enter name, name2, age, rate: ");
scanf("%s %s %d %f",name,name2,age, rate);

while( strcmp(name, "xxx") )
{    ……
}
```

segmentation fault
core dump

segmentation fault
core dump

117

117

Whenever you need to set a pointer's pointee

e.g.,

- `*ptr = var;`
- `scanf("%s", ptr);`
- `strcpy(ptr, "hello");`
- `fgets(ptr, 10, STDIN);`
- ……
- `*ptrArr[2] = var;` `// pointer array`

Ask yourself: Have you done one of the following

1. `ptr = &var.` `/* direct */`
   `arr[20]; ptr=&arr[0];`
2. `ptr = ptr2` `/* indirect, assuming ptr2 is good */`
3. `ptr = (..)malloc(....)` `/* later */`

118

118

Pointers K&R Ch 5

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

Today

- Array of pointers (5.6-5.9)
- Command line arguments (5.10)

Next

- Memory allocation (extra)
- Pointer to structures (6.4)
- Pointer to functions

YORK UNIVERSITÉ UNIVERSITY

119