Pointers K&R Ch 5

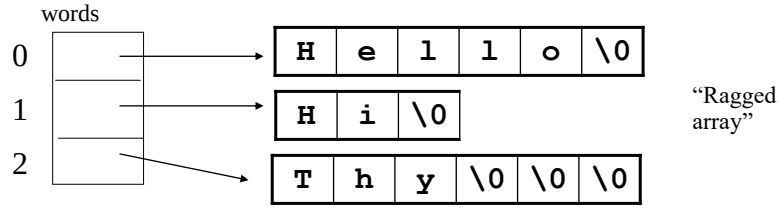
- Basics: Declaration and assignment (5.1)
- Pointer to Pointer (5.6)
- Pointer and functions (pass pointer by value) (5.2)
- Pointer arithmetic +- ++ -- (5.4)
- Pointers and arrays (5.3)
  - Stored consecutively
  - Pointer to array elements p + i = &a[i] *(p+i) = a[i]
  - Array name contains address of 1st element a = &a[0]
  - Pointer arithmetic on array (extension) p1-p2 p1<>!= p2
  - Array as function argument – "decay"
  - Pass sub_array
- Array of pointers (5.6)
- Pointer arrays vs. two dimensional arrays (5.9)
- Command line argument (5.10)
- Memory allocation (extra)
- Pointer to structures (6.4)
- Pointer to functions

YORK UNIVERSITÉ UNIVERSITY

Array of pointers to strings

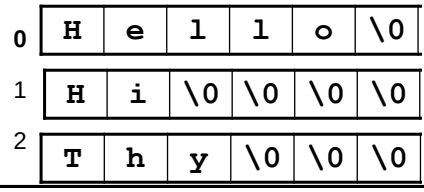
Advantage of Pointer Arrays (vs. 2D array)

```
char * words[]={"Hello", "Hi", "thy"};
```



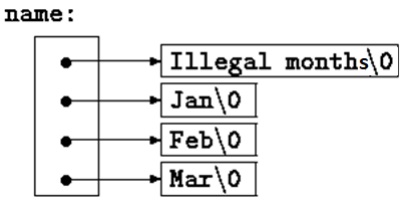
Both store table of strings. What is the difference?

```
char words[3][6] = {"Hello", "Hi", "Thy"};
```



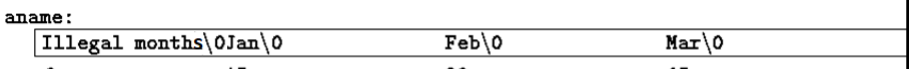
Advantage of Pointer Arrays (vs. 2D array) example 2

```
char *name[]={"Illegal months, "Jan", "Feb", "Mar"};
```



"Ragged array"

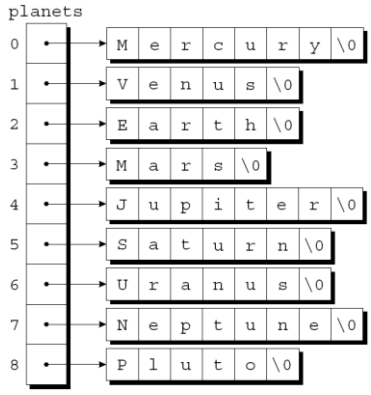
```
char aname[][15]={"Illegal months","Jan","Feb","Mar"};
```

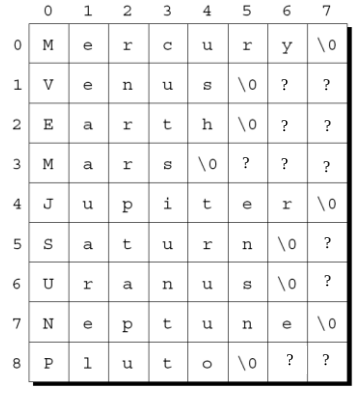


108

Advantage of Pointer Arrays (vs. 2D array) example 3

```
char planets[][8] = {"Mercury", "Venus", "Earth",
                     "Mars", "Jupiter", "Saturn",
                     "Uranus", "Neptune", "Pluto"};
```





	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	?	?
2	E	a	r	t	h	\0	?	?
3	M	a	r	s	\0	?	?	?
4	J	u	p	i	t	e	r	\0
5	S	a	t	u	r	n	\0	?
6	U	r	a	n	u	s	\0	?
7	N	e	p	t	u	n	e	\0
8	P	l	u	t	o	\0	?	?

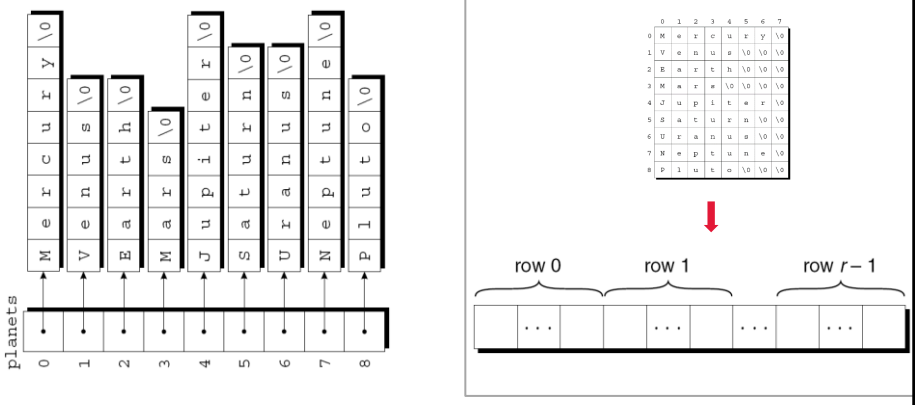
```
char *planets[] = {"Mercury", "Venus", "Earth",
                   "Mars", "Jupiter", "Saturn",
                   "Uranus", "Neptune", "Pluto"};
```

109

109

Advantage of Pointer Arrays (vs. 2D array) example 3

```
char planets[][8] = {"Mercury", "Venus", "Earth",
                     "Mars", "Jupiter", "Saturn",
                     "Uranus", "Neptune", "Pluto"};
```

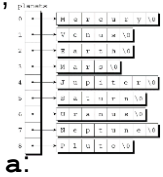



```
char *planets[] = {"Mercury", "Venus", "Earth",
                   "Mars", "Jupiter", "Saturn",
                   "Uranus", "Neptune", "Pluto"};
```


Advantage of Pointer Arrays (vs. 2D array)

```
int a[10][20];
int *b[10];
```

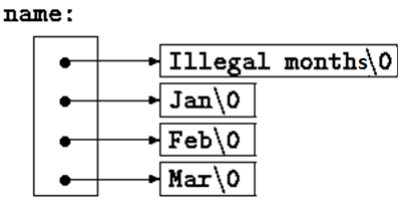
- **a**: 200 int-sized locations have been set aside.
  - Total size: 10*20*4
- **b**: only 10 pointers are allocated (and not initialized); initialization must be done explicitly.
  - Total size: 10*8 + size of all pointees



- Potential advantage of pointer array **b** vs. 2D array **a**:
  1. the rows of the array may be of different lengths (potentially saving space).
  2. Another advantage? Swap rows!

Advantage of Pointer Arrays (vs. 2D array)

```
char *name[] = {"Illegal  months", "Jan","Feb","Mar"};
```



How to swap "rows"?

sizeof name: 4*8 = 32 total memory size 4*8+15+4+4+4=59

```
char aname[][15]={"Illegal months","Jan","Feb","Mar"};
```

aname:

Illegal months\0	Jan\0	Feb\0	Mar\0
0	15	30	45

112 sizeof aname: 4*15 = 60

How to swap rows?

112

```
char planets[][8] = {"Mercury", "Venus", "Earth",
                     "Mars", "Jupiter", "Saturn",
                     "Uranus", "Neptune", "Pluto"};
```

	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	?	?
2	E	a	r	t	h	\0	?	?
3	M	a	r	s	\0	\0	\0	?
4	J	u	p	i	t	e	r	\0
5	S	a	t	u	r	n	\0	?
6	U	r	a	n	u	s	\0	?
7	N	e	p	t	u	n	e	\0
8	P	l	u	t	o	\0	?	?

```
for(i=0; i<9; i++)
  printf("%s",arr[i]);
```

```
"Mercury"
"Venus"
"Earth"
"Mars"
...
```

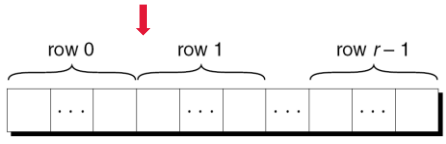
```
"Venus"
"Mercury"
"Earth"
"Mars"
..."
```

Sort 

113

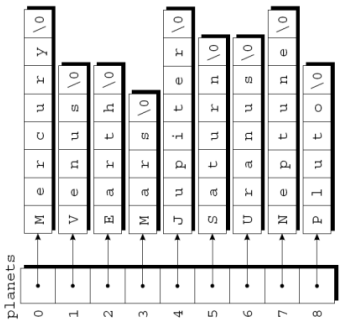
113

```
char planets[][8] = {"Mercury", "Venus", "Earth",
                     "Mars", "Jupiter", "Saturn",
                     "Uranus", "Neptune", "Pluto"};
```



How to swap 1st 2nd row?

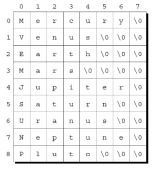
```
char tmp[8];
tmp = planets[0]
planets[0] = planets[1]
planets[1] = tmp;
```

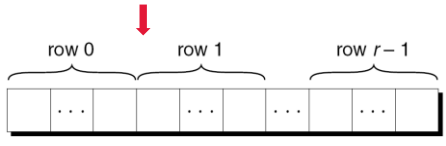



```
char *planets[] =
{"Mercury", "Venus", "Earth",
   "Mars", "Jupiter", "Saturn",
  "Uranus", "Neptune", "Pluto"};
```



```
char planets[][8] = {"Mercury", "Venus", "Earth",
                     "Mars", "Jupiter", "Saturn",
                     "Uranus", "Neptune", "Pluto"};
```



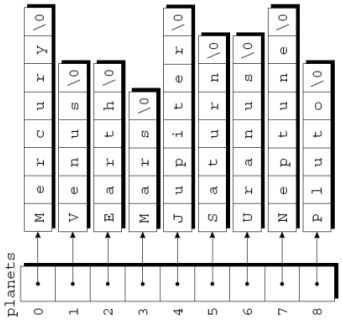


How to swap 1st 2nd row?

~~`char tmp[8];`~~
~~`tmp = planets[0]`~~
~~`planets[0] = planets[1]`~~
~~`planets[1] = tmp;`~~

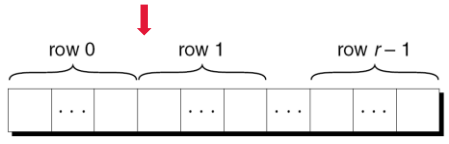
```
strcpy(tmp,planets[0]);
strcpy(planets[0],planets[1]);
strcpy(planets[1],tmp);
```

O(n)




```
char *planets[] =
{"Mercury", "Venus", "Earth",
   "Mars", "Jupiter", "Saturn",
  "Uranus", "Neptune", "Pluto"};
```

```
char planets[][8] = {"Mercury", "Venus", "Earth",
                     "Mars", "Jupiter", "Saturn",
                     "Uranus", "Neptune", "Pluto"};
```

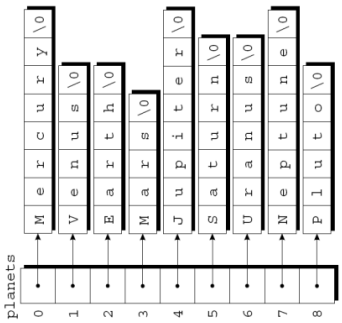


How to swap 1st 2nd row?

```
char tmp[8];
tmp = planets[0]
planets[0] = planets[1]
planets[1] = tmp;
```

```
for(i=0;i<8;i++){  //copy char one by one
  char tmp = planets[0][i];
  planets[1][i] = planets[0][i];
  planets[0][i] = tmp;
}
```

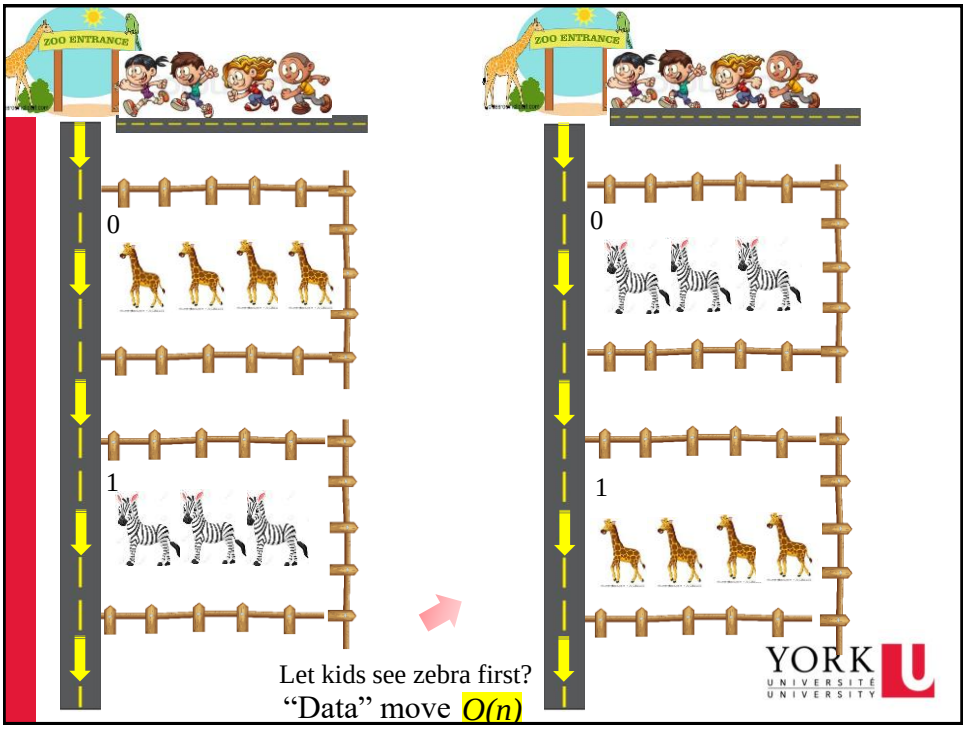
O(n)




```
char *planets[] =
{"Mercury", "Venus", "Earth",
   "Mars", "Jupiter", "Saturn",
  "Uranus", "Neptune", "Pluto"};
```

Stop here

116

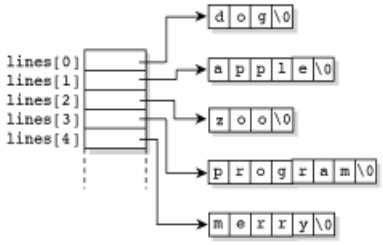


Let kids see zebra first?
"Data" move *O(n)*

117

Efficient manipulation of strings

```
char *lines[]={"dog", "apple", "zoo", "program", "merry"};
// swap data [0] vs [1]
```




```
for(i=0; i<5; i++)
  printf("%s",lines[i]);

"dog"
"apple"
"zoo"
"program"
"merry"

"apple"
"dog"
"zoo"
"program"
"merry"
```

Sort 

118

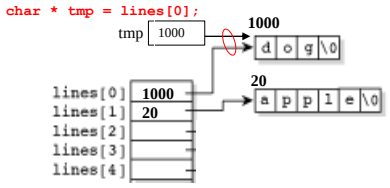
Efficient manipulation of strings

```
char *lines[]={"dog", "apple", "zoo", "program", "merry"};
// swap data [0] vs [1]
char * tmp = lines[0]; // tmp gets 1000, pointing to "dog"
lines[0] = lines[1];   // [0] gets 20, pointing to "apple"
lines[1] = tmp;        // [1] gets 1000, pointing to "dog"
```

119

Efficient manipulation of strings

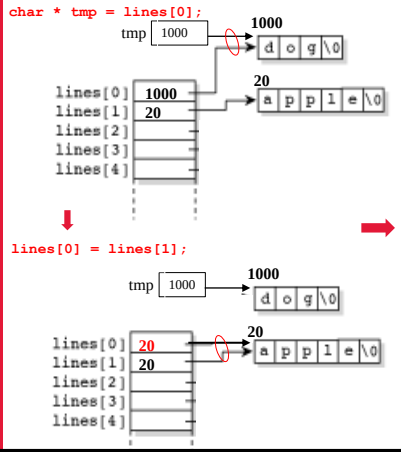
```
char *lines[]={"dog", "apple", "zoo", "program", "merry"};
// swap data [0] vs [1]
char * tmp = lines[0]; // tmp gets 1000, pointing to "dog"
lines[0] = lines[1];   // [0] gets 20, pointing to "apple"
lines[1] = tmp;        // [1] gets 1000, pointing to "dog"
```



120

Efficient manipulation of strings

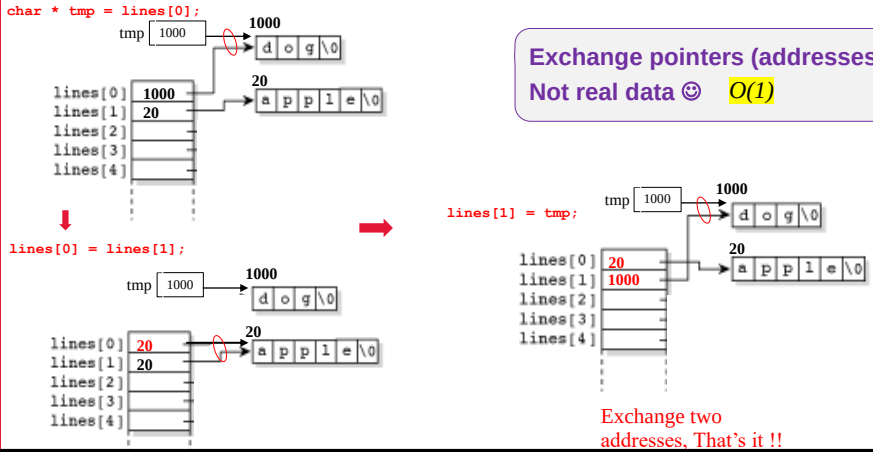
```
char *lines[]={"dog", "apple", "zoo", "program", "merry"};
// swap data [0] vs [1]
char * tmp = lines[0]; // tmp gets 1000, pointing to "dog"
lines[0] = lines[1];   // [0] gets 20, pointing to "apple"
lines[1] = tmp;        // [1] gets 1000, pointing to "dog"
```



121

Efficient manipulation of strings

```
char *lines[]={"dog", "apple", "zoo", "program", "merry"};
// swap data [0] vs [1]
char * tmp = lines[0]; // tmp gets 1000, pointing to "dog"
lines[0] = lines[1];   // [0] gets 20, pointing to "apple"
lines[1] = tmp;        // [1] gets 1000, pointing to "dog"
```

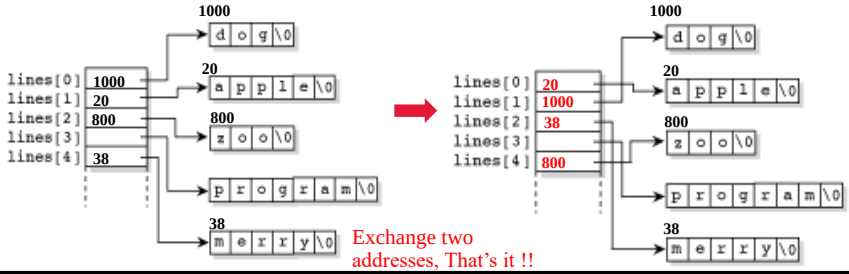


122

Efficient manipulation of strings

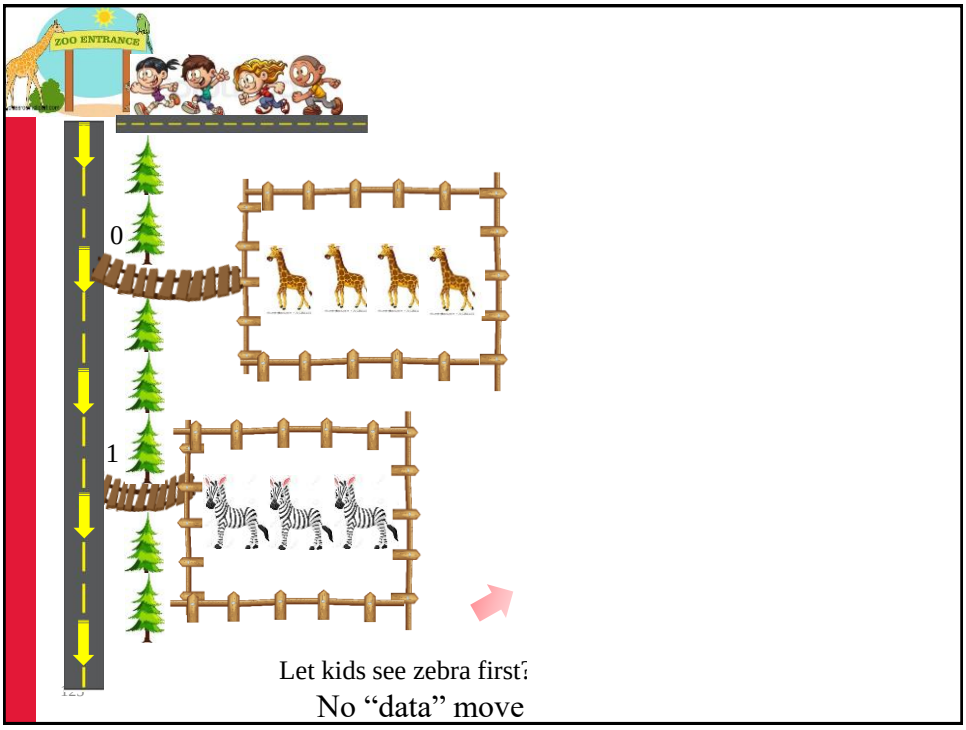
```
char *lines[]={"dog", "apple", "zoo", "program", "merry"};
// [0] vs [1]
char * tmp = lines[0]; // tmp gets 1000, pointing to "dog"
lines[0] = lines[1];   // [0] gets 20, pointing to "apple"
lines[1] = tmp;        // [1] gets 1000, pointing to "dog"

// [2] vs.[4]
tmp = lines[2];       // tmp points to "zoo"
lines[2] = lines[4]; // [2] points to "merry"
lines[4] = tmp;       // [4] points to "zoo"
```

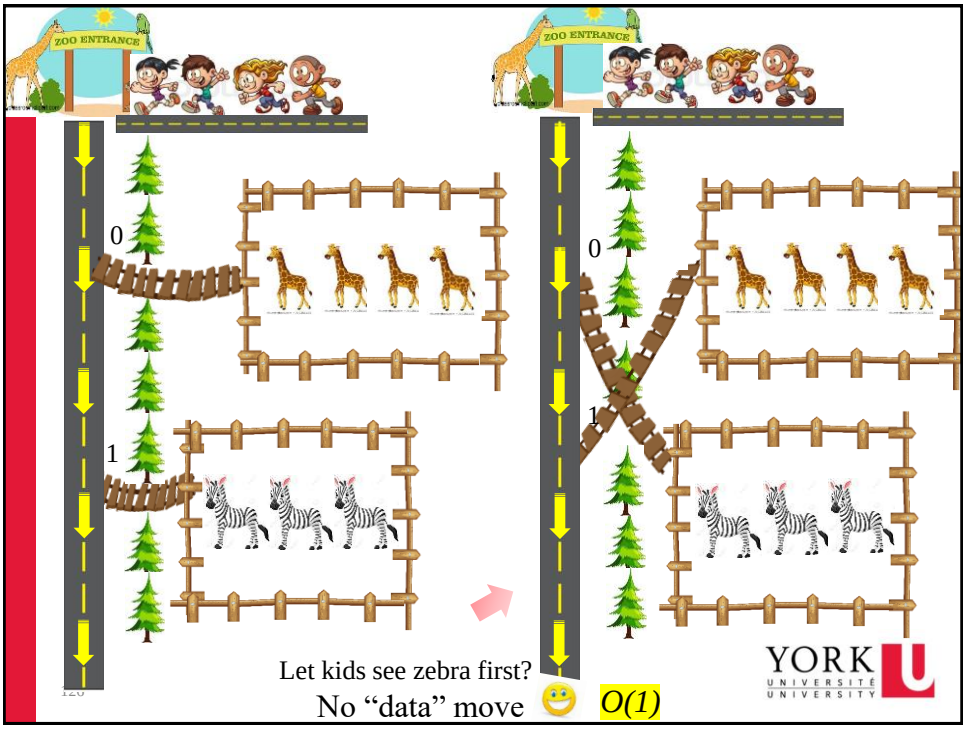


12

123


ZOO ENTRANCE
0
1
Let kids see zebra first?
No “data” move

125

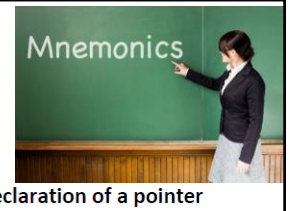

ZOO ENTRANCE
ZOO ENTRANCE
0
1
0
1
Let kids see zebra first?
No “data” move
O(1)
YORK
UNIVERSITÉ
UNIVERSITY

126

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
- Array of pointers (5.6)
- Pointer arrays vs. two dimensional arrays (5.9)
- Command line argument (5.10)
- Memory allocation (extra)
- Pointer to structures (6.4)
- Pointer to functions

YORK UNIVERSITÉ UNIVERSITY



As mentioned in the textbook and class, the declaration of a pointer related variable is intended as a *mnemonic* (means 'it helps you memorize things'). For example, declaration `int *ptr;` can be interpreted as "expression `*ptr` is an `int`" -- thus `ptr` is an integer pointer.

Following this rule, what is the type that `argv` is declared to be?

```
int main(int argc, char *argv[])
{……}
```

YORK UNIVERSITÉ UNIVERSITY

Command-Line Arguments (5.10)
(Program arguments)

- Up to now, program interacts with user (i.e., receives user input) from stdin, using **`getchar, scanf`**, **`fgets`**, etc.
- Are there other ways?
- Program can take arguments.

```
int main(int argc, char *argv[])
```



```
public static void main(String[] args)
```

Command-Line Arguments (5.10)
(Program arguments)

- Up to now, we defines main as **`int main()`**
- Usually it is defined as

```
int main(int argc, char *argv[])
```

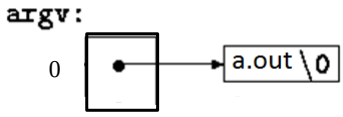
  - **`argc`** is the number of arguments (including program name)
  - **`argv`** is an array containing the arguments.
  - **`argv[0]`** is a pointer to a string with the program name. So **`argc`** is at least 1. (Java?)
  - **`Optional arguments:`** **`argv[1] ~~ argv[argc-1]`**

Can any name

Command-line arguments (program arguments)

- red 421 % **a.out**
  argv[0]: a.out

  argc: 1

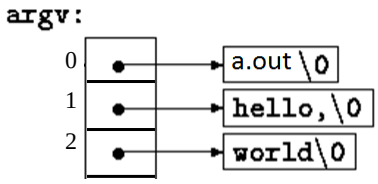


1 arg

- red 421 % **a.out hello, world**
  argv[0]: a.out
  argv[1]: hello,
  argv[2]: world optional

  argc: 3

3 args



133

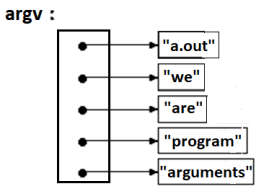
```
public static void main(String[] args)
```

Different from Java

No argc why?

- red 421 % **a.out we are program arguments**
  argv[0]: a.out 0 1 2 3 4
  argv[1]: we
  argv[2]: are
  argv[3]: program
  argv[4]: arguments

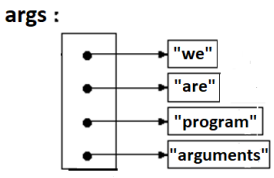
  argc: 5



5 args

- red 422 % **java Prog we are program arguments**
  args[0]: we 0 1 2 3
  args[1]: are
  args[2]: program
  args[3]: arguments

  args.length: 4



4 args

134

Command-Line Arguments (cont.)

file.c

```
main( int argc, char *argv[] ) {
   int i;
   printf("Number of arg: %d\n",  ? );
   for(i=0; i<argc; i++ )
     printf("argv[%d]: %s\n",i,  ? );
}
```

```
% gcc file.c
% a.out
Number of arg: 1
argv[0]: a.out
```

```
% gcc file.c -o xyz
% xyz how are you
Number of arg: 4
argv[0]: xyz
argv[1]: how
argv[2]: are
argv[3]: you
```

```
% a.out how "are you"
Number of arg: 3
argv[0]: a.out
argv[1]: how
argv[2]: are you
```

For your information

Command-Line Arguments (cont.)

file.c

```
main( int argc, char *argv[] ) {
   int i;
   printf("Number of arg: %d\n", argc );
   for(i=0; i<argc; i++ )
     printf("argv[%d]: %s\n",i, argv[i] );
}
```

*(argv+i) // compiler

```
% gcc file.c
% a.out
Number of arg: 1
argv[0]: a.out
```

```
% gcc file.c -o xyz
% xyz how are you
Number of arg: 4
argv[0]: xyz
argv[1]: how
argv[2]: are
argv[3]: you
```

```
% a.out how "are you"
Number of arg: 3
argv[0]: a.out
argv[1]: how
argv[2]: are you
```

For your information

Command-Line Arguments (cont.)

file.c

```
main( int argc, char *argv[] ) {
    int i;
    printf("Number of arg: %d\n", argc );
    char ** p = argv; // &argv[0]
    for(i=0; i<argc; i++ )
      printf("argv[%d]: %s\n", i,        );
}
```

```
% gcc file.c
% a.out
Number of arg: 1
argv[0]: a.out
```

```
% gcc file.c -o xyz
% xyz how are you
Number of arg: 4
argv[0]: xyz
argv[1]: how
argv[2]: are
argv[3]: you
```

```
% a.out how "are you"
Number of arg: 3
argv[0]: a.out
argv[1]: how
argv[2]: are you
```

For your information

137

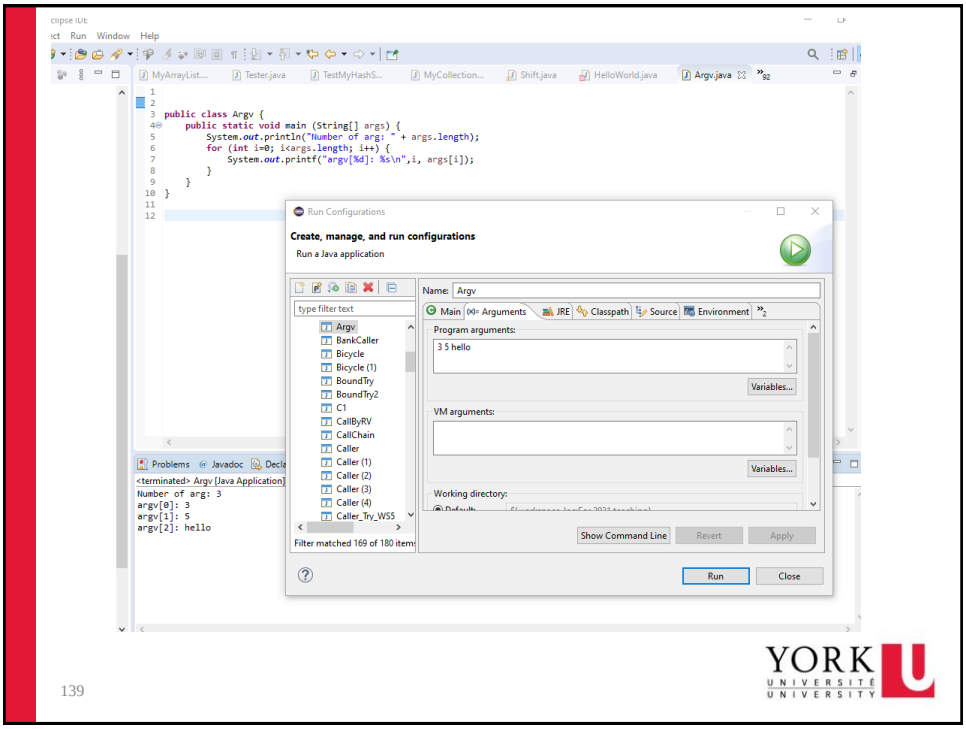
137

```
indigo 52 % a.out
Number of arg: 1
argv[0]: a.out
indigo 53 %
indigo 53 % a.out hi
Number of arg: 2
argv[0]: a.out
argv[1]: hi
indigo 54 %
indigo 54 % a.out hello the world
Number of arg: 4
argv[0]: a.out
argv[1]: hello
argv[2]: the
argv[3]: world
indigo 55 %
```

```
indigo 64 % javac argv.java
indigo 65 % java argv
Number of arg: 0
indigo 66 %
indigo 66 % java argv hi
Number of arg: 1
argv[0]: hi
indigo 67 %
indigo 67 % java argv hello the world
Number of arg: 3
argv[0]: hello
argv[1]: the
argv[2]: world
indigo 68 %
```

138

138



Command-Line Arguments (cont.)

argvSum.c

```
main( int argc, char *argv[] ) {
    int i; int sum;
    for(i=1; i<argc; i++ )
      sum +=
    printf("Sum is %d\n", sum );
}
```

```
% gcc argvSum.c          % gcc argvSum.c -o xyz
% a.out                  % xyz 1 3 5
 Sum is 0                 Sum is 9

% a.out 3 5              % xyz 10 20 33
 Sum is 8                 Sum is 63
```

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
- Array of pointers (5.6)
- Pointer arrays vs. two dimensional arrays (5.9)
- Command line argument (5.10)

today

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

YORK UNIVERSITÉ UNIVERSITY