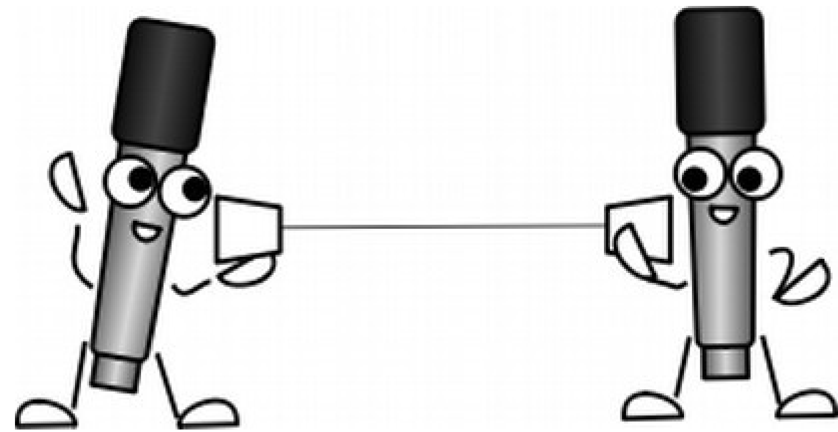
CS 261
Fall 2016

Mike Lam, Professor



Strings and I/O

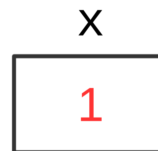
Warm-Up: Pointers

```
int x = 1;

int y[4] = {2, 3, 4, 5};

int *p = &x;

*p = 6;

p = y;

*p = 7;
```

What are the values of x and y at the end?

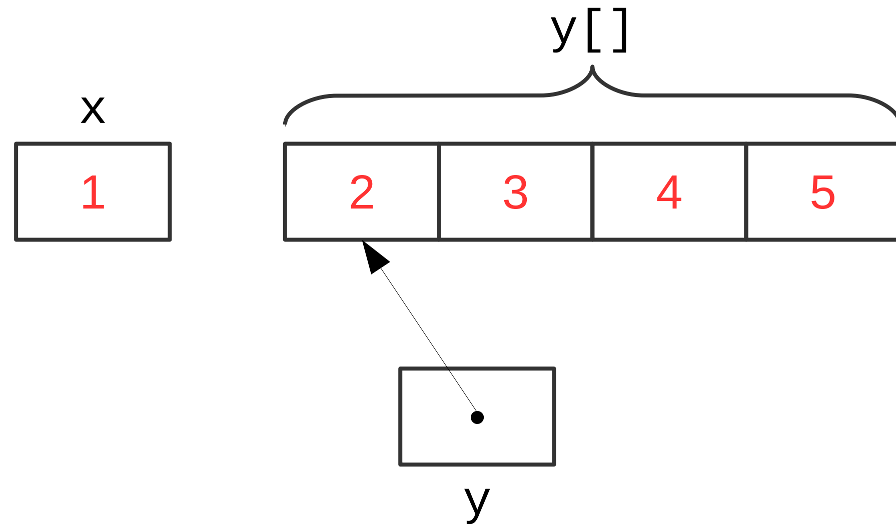
Pointers

```
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
```



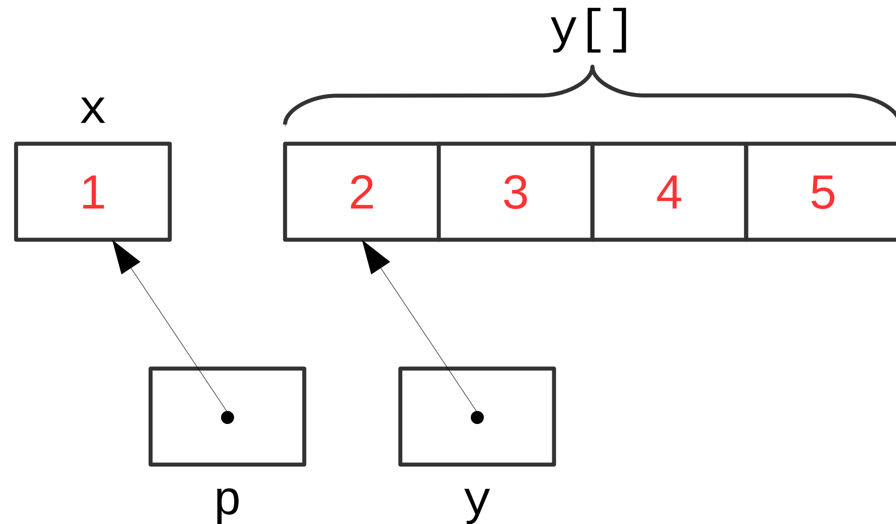
Pointers

```
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
```



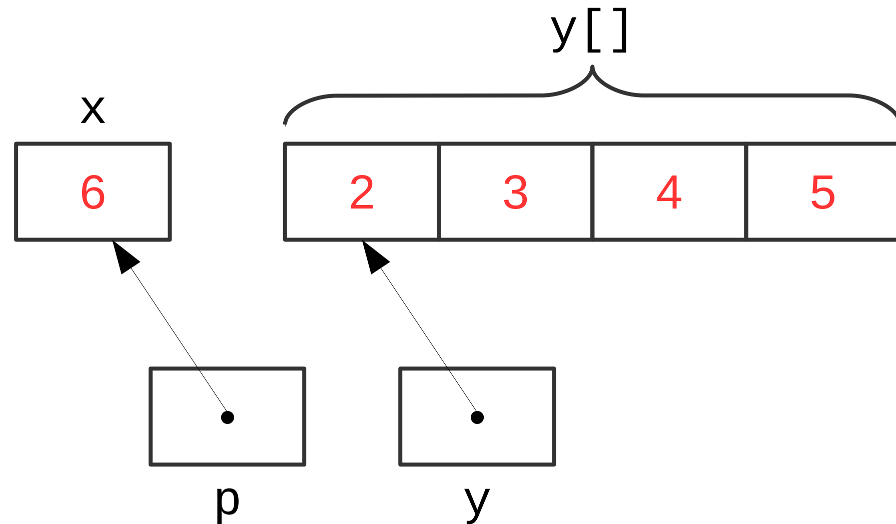
Pointers

```
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
```



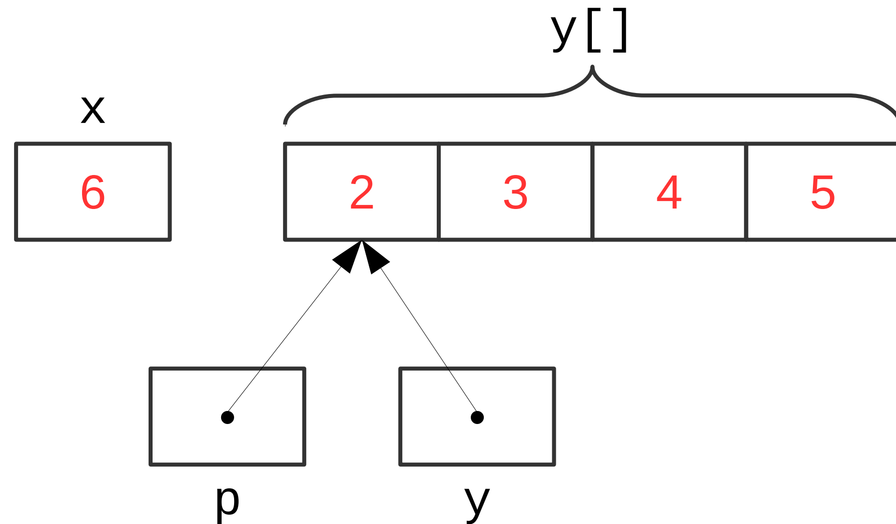
Pointers

```
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
```



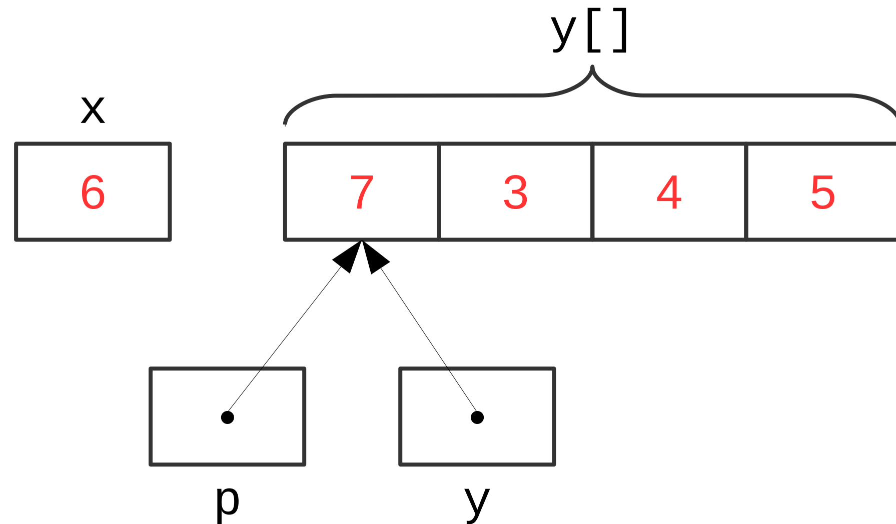
Pointers

```
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
```



Pointers

```
int x = 1;
int y[4] = {2, 3, 4, 5};
int *p = &x;
*p = 6;
p = y;
*p = 7;
```



What will this C code print?

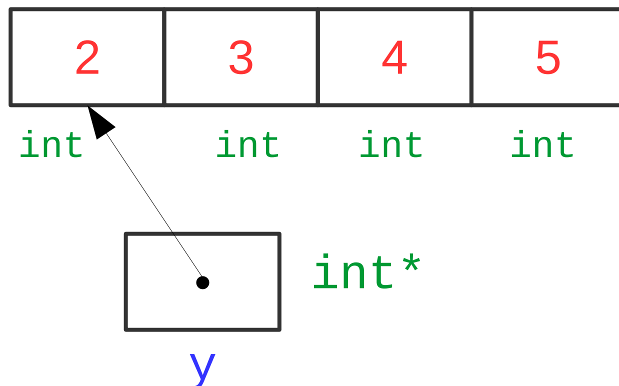
```
int a = 42;
int b = 7;
int c = 999;
int *t = &a;
int *u = NULL;
printf("%d %d\n", a, *t);

c = b;
u = t;
printf("%d %d\n", c, *u);

a = 8;
b = 8;
printf("%d %d %d %d\n", b, c, *t, *u);

*t = 123;
printf("%d %d %d %d %d\n", a, b, c, *t, *u);
```

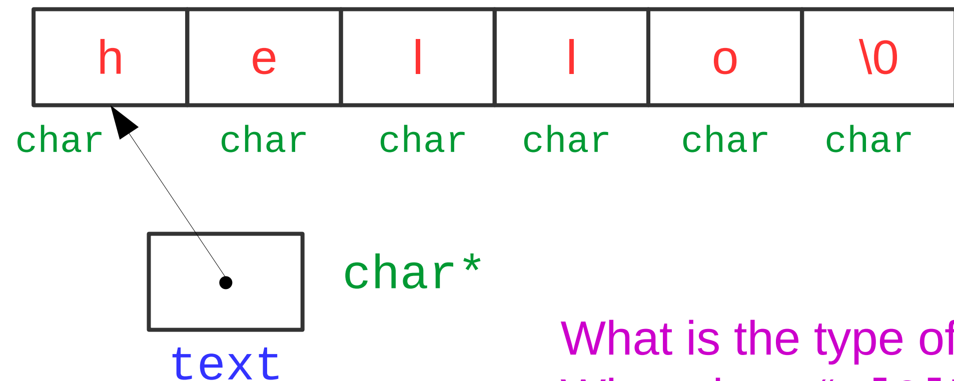
Arrays and Pointers

- In C, array names are just pointers!

```
int y[] = {2, 3, 4, 5};
```

- Same goes for strings (arrays of chars)
  - `char text[] = “hello”;`
- Indexing and dereferencing pointers are equivalent
  - `*y ≡ y[0]`　　　`*(y+1) ≡ y[1]`





What is the type of “`*y`”?
What about “`y[0]`”?
What about “`y[4]`”?

Questions to Ask

- What is the type of this variable?
  - If it’s a pointer, what does it point to?
- Where is this variable located?
  - Usually: static region, stack, or heap
  - If it’s a pointer, where’s the thing it’s pointing to?
- How large (in bytes) is the variable?
  - If it’s a pointer to an array, how large is the array?

“What? Where? How big?”

C Strings

- Strings are arrays of chars terminated (by convention) with null char ('\0')
  - Declare and initialize (static/stack, no explicit size needed):
    - `char *name = “John Smith”;`
  - Declare only (static/stack, size needed):
    - `char name[11];`
  - Declare only (heap, size needed):
    - `char *name = (char*) malloc (sizeof(char) * 11);`

- Useful functions (may need to #include <string.h>)
  - Find length: `strlen`
  - Copy string or convert / format data into string: `snprintf`
  - Convert to long / float: `strtol` / `strtof`
  - Compare strings: `strncmp`
  - Search for substring: `strstr`

Modified “Hello, World”

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

#define STR_LEN 8

int main(int argc, char **argv)
{
    // check parameters
    if (argc != 3) {
        fprintf(stderr, "Usage: ./hello2 <fname> <lname>\n");
        exit(EXIT_FAILURE);
    }

    // convert name to "First L." format
    char fullname[STR_LEN];
    snprintf(fullname, STR_LEN, "%s %c.", argv[1], argv[2][0]);

    // output new full name
    printf("Hello, %s!\n", fullname);

    return EXIT_SUCCESS;
}
```

Information = Bits + Context

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Standard I/O

- Buffered streams: `stdin`, `stdout`, `stderr` (type is `FILE*`)
  - Flushed when newline (‘`\n`’) encountered
  - Use CTRL-D to indicate end-of-file when typing input from the terminal
- Formatted input/output (`scanf` / `printf`)
  - Variable number of arguments (varargs)
  - Format string and type specifiers:
    - `%d` for signed int, `%u` for unsigned int
    - `%c` for chars, `%s` for strings (arrays of chars)
    - `%f` or `%e` for float, `%x` for hex, `%p` for pointer
    - Prepend ‘`l`’ for long (e.g., `%ld` = long signed int or `%lx` = long hex)
    - Include number for fixed-width field (e.g., `%20s` for a 20-character field)
- Input: beware of buffer overruns
  - Declare a fixed-size buffer and use “safe” input functions (e.g., `fgets`)
  - You may NOT use unsafe functions in this course! (e.g., `gets`)

File I/O

- *Opaque* file/stream handles: `FILE*`
- Open a file: `fopen`
  - Mode: read (‘`r`’), write (‘`w`’), append (‘a’)
- Read a character: `fgetc`
- Read a line of text: `fgets`
- Read binary data: `fread`
- Set current file position: `fseek`
- Write formatted text: `fprintf`
- Write binary data: `fwrite`
- Close a file: `fclose`

Documentation

fgets

Defined in header <stdio.h>

```
char *fgets( char          *str, int count, FILE          *stream );   (until C99)
char *fgets( char *restrict str, int count, FILE *restrict stream );   (since C99)
```

Reads at most `count - 1` characters from the given file stream and stores them in the character array pointed to by `str`. Parsing stops if end-of-file occurs or a newline character is found, in which case `str` will contain that newline character. If no errors occur, writes a null character at the position immediately after the last character written to `str`.

The behavior is undefined if `count` is less than `1`.

Parameters

str - pointer to an element of a char array

count - maximum number of characters to write (typically the length of `str`)

stream - file stream to read the data from

Return value

`str` on success, null pointer on failure.

If the failure has been caused by end-of-file condition, additionally sets the *eof* indicator (see `feof()`) on `stream`. The contents of the array pointed to by `str` are not altered in this case.

If the failure has been caused by some other error, sets the *error* indicator (see `ferror()`) on `stream`. The contents of the array pointed to by `str` are indeterminate (it may not even be null-terminated).

Simple “cat” program

```c
#include <stdio.h>

#define BUF_SIZE 1024

int main (int argc, char **argv)
{
    char buffer[BUF_SIZE];
    while (fgets(buffer, BUF_SIZE, stdin) != NULL) {
        printf("%s", buffer);
    }
    return 0;
}
```

Exercise

- Write a program (`rev.c`) that reverses every line of an input file

  1) First, just accept input via standard in (`stdin`)

     `./rev <input.txt` (or just `./rev` and type text followed by CTRL-D)

  2) Then, allow the user to specify the filename on the command line

     `./rev input.txt`

Hint: use `fgets()` to read the input a line at a time into a char array, printing the characters in reverse after reading each line

`FILE* fopen (char *filename, char *mode)`
Open a file (mode: 'r' for read, 'w' for write, 'a' for append)

`char* fgets (char *str, int count, FILE *stream)`
Read a line of text input from a file (returns str, count is max chars)

`size_t strlen (char *str)`
Calculate the length of a null-terminated string

Sample input:

```
Hello, world!
My name is Bob.

ENOD
```

Sample output:

```
!dlrow ,olleH
.boB si eman yM

DONE
```