CMPE-013/L

Introduction to "C" Programming

Max Lichtenstein

Piazza Poll: What happens when we try to run this code?

```
float * foo = NULL;
*foo = 77;
printf("foo = %x", foo);
```

A) It prints "77"

B) It prints some address

C) Runtime error

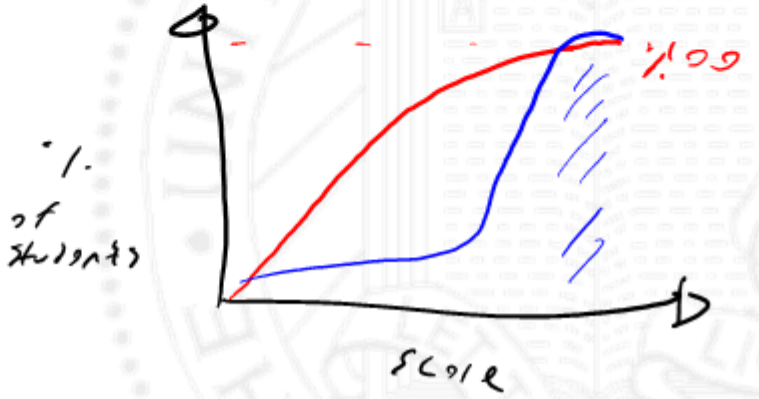
D) Doesn't even compile

Roadmap

- Announcements, Lab 4 debrief
- Operators
- Pointers, again
  - Some examples
  - Common pointer mistakes
- Break
- Heap, malloc(), free()
- Lab5 :
  - Linked Lists
  - String comparison

Announcements

- Lab 5 is out
  - Warning: Hard lab
- Withdraw Deadline is Friday
- Your progress



Lab 4 Debrief

Undefined reference

```
int ProcessBackspaces(
    return 0;
}
```

Operators

Operators

- Operators join constants and variables
- Have an order of precedence and associativity
- LOTS of operators in C

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

precedence

PEMDAS

Max Lichtenstein

precedence

Category	Operator
Postfix	() [] -> . ++ - -
Unary	+ - ! ~ ++ - - (type) * & sizeof
Multiplicative	* / %
Additive	+ -
Shift	<< >>
Relational	< <= > >=
Equality	== !=
Bitwise AND	&
Bitwise XOR	^
Bitwise OR	|
Logical AND	&&
Logical OR	||
Conditional	?:
Assignment	= += -= *= /= %= >>= <<= &= ^= |=
Comma	,

associativity

int matrix[4][4];

matrix[1][2] = 12;

matrix([4][4])

(matrix[4])[4] ✓

((1 << 100) >> 100)

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left ✓
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right ✓

Assignment

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Max Lichtenstein

Practice

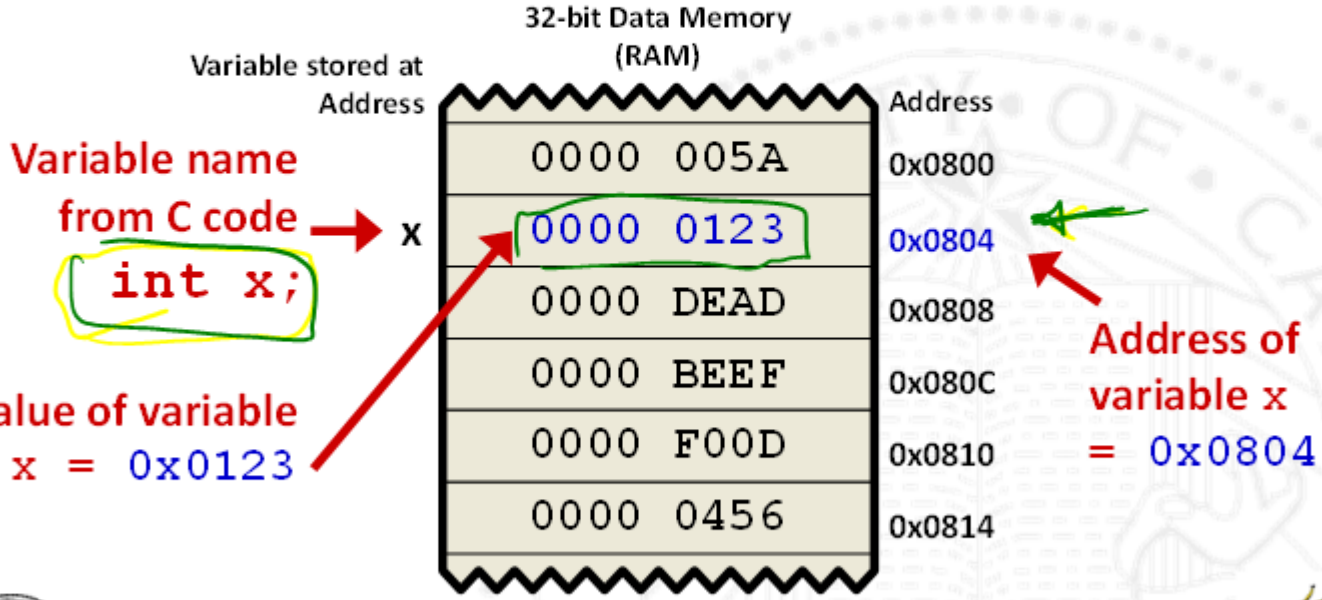
```
int x, y, z;
int array[3] = {0x1, 0x20, 0x300};
int * arr = array;

x = (arr++)[2];

y = (arr[2])++;

z = (x++) + (y * (*(arr--)));

printf("x,y,z = %d, %d, %d\n", x, y, z);
```

Max Lichtenstein

Pointers Review

Pointers

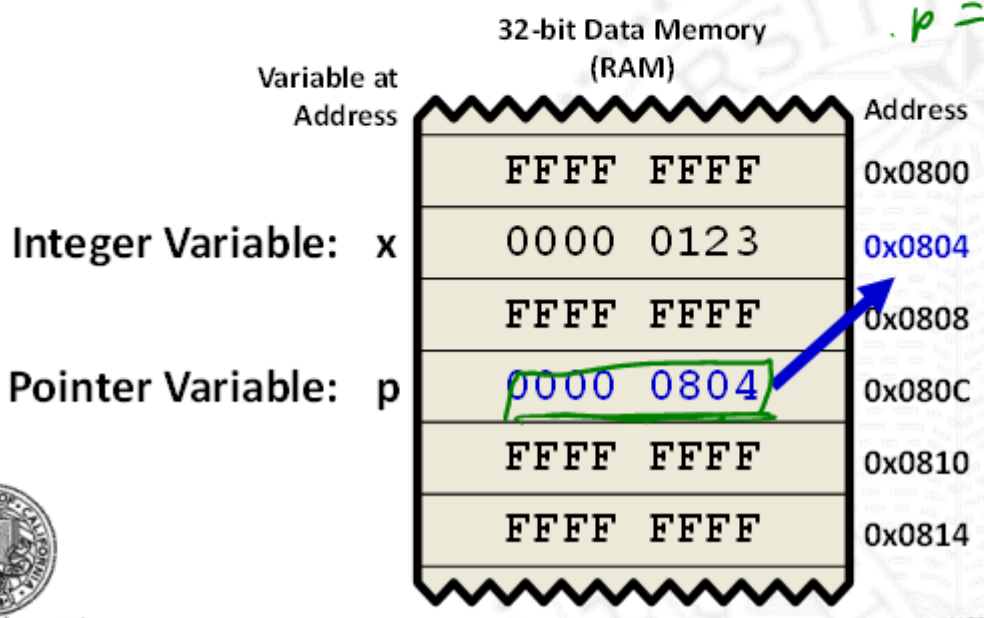
Address versus value



Pointers

What are pointers?

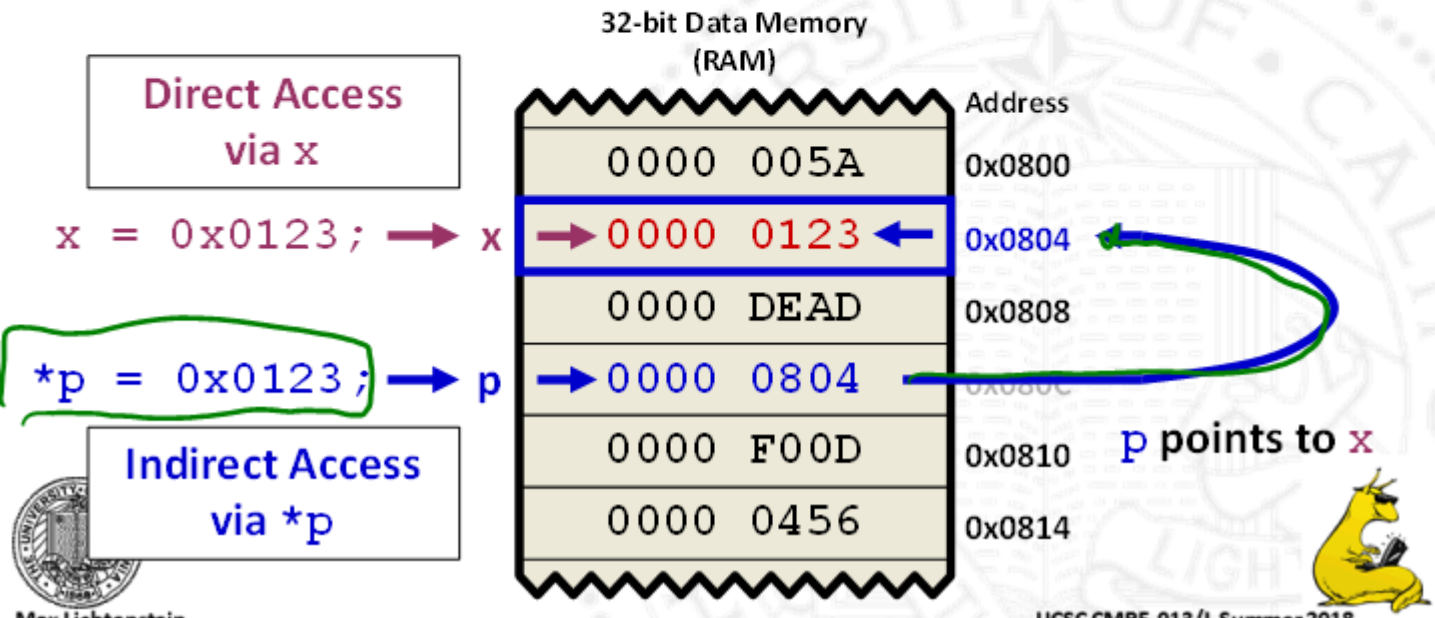
- A pointer holds the address of another variable or function



Pointers

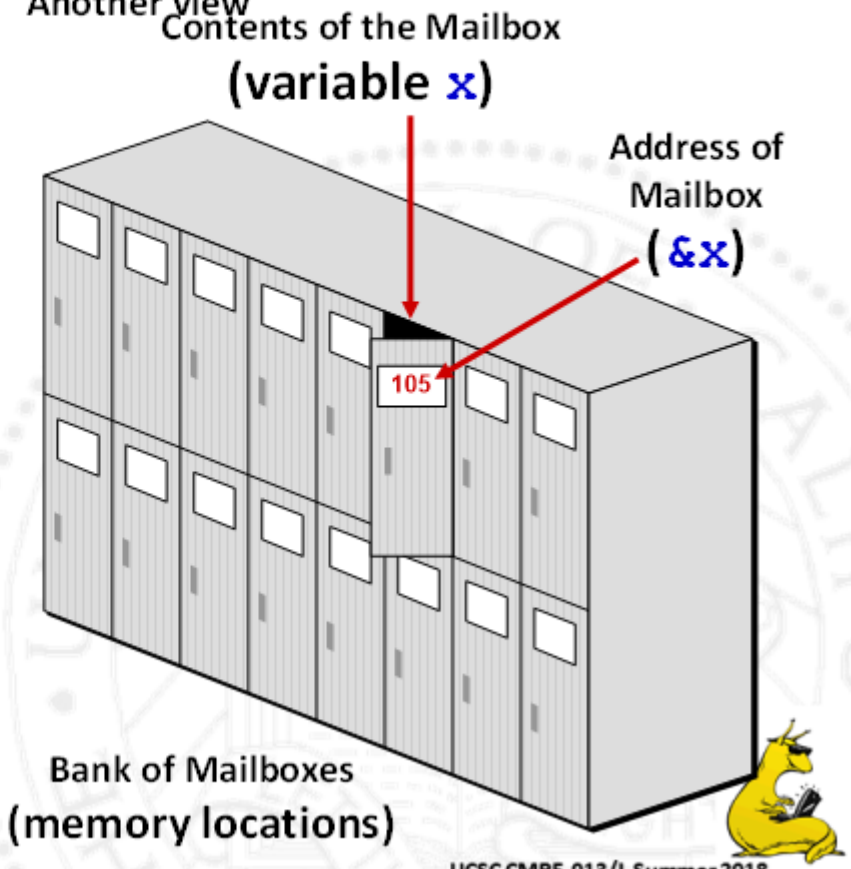
What do they do?

- A pointer allows us to indirectly access a variable (just like indirect addressing in assembly language)



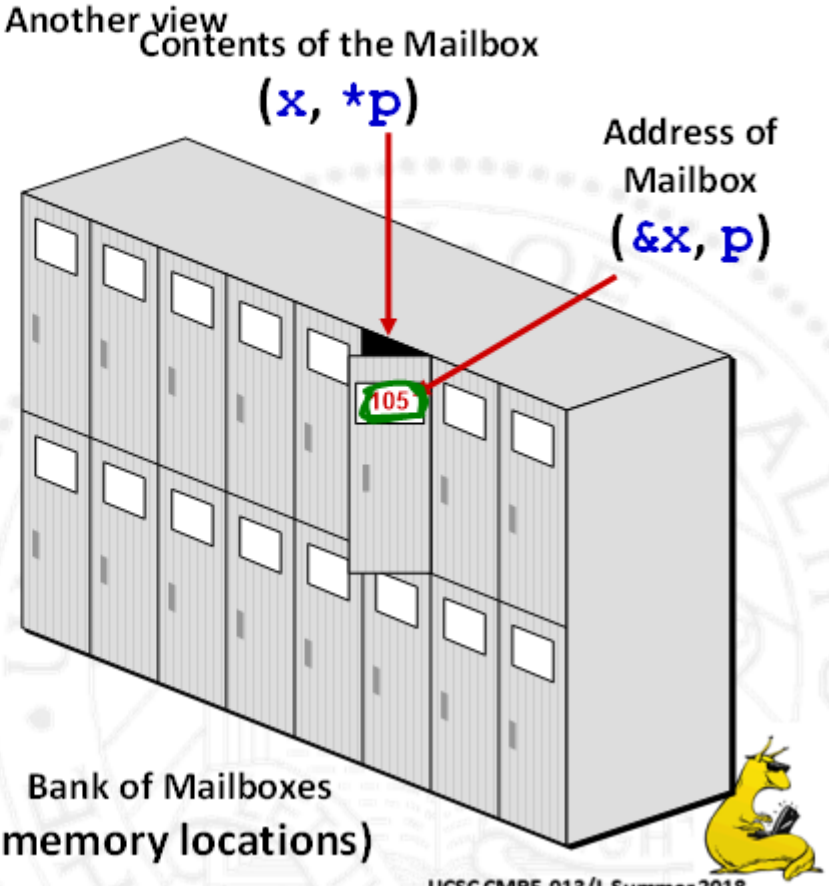
Pointers

Another view

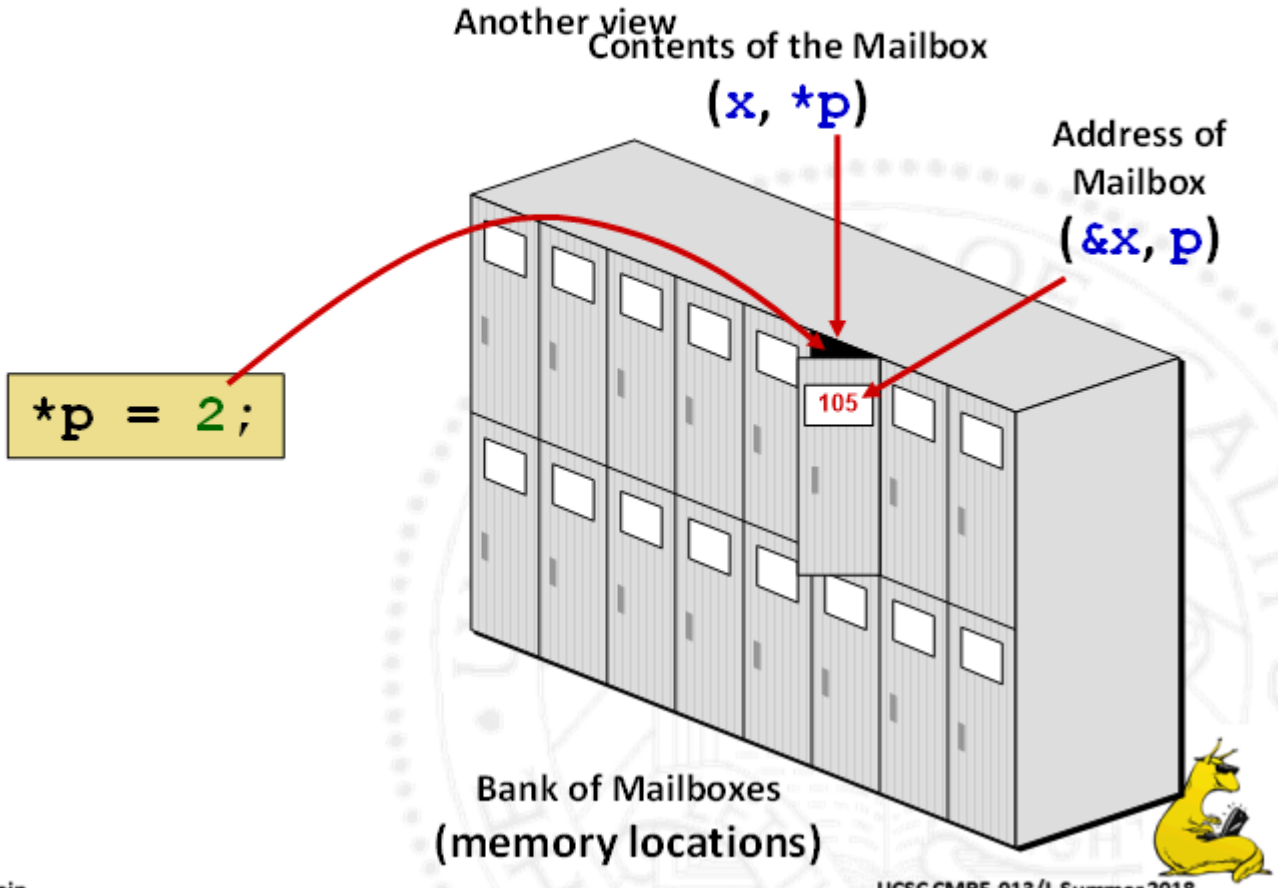


Pointers

```
p = &x;
```


Another view
Contents of the Mailbox
(x, *p)
Address of
Mailbox
(&x, p)
105
Bank of Mailboxes
(memory locations)

Pointers

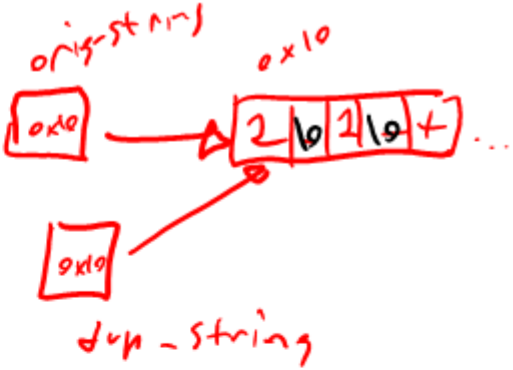

Another view
Contents of the Mailbox
(x, *p)
Address of Mailbox
(&x, p)
*p = 2;
105
Bank of Mailboxes
(memory locations)

Pointer Examples

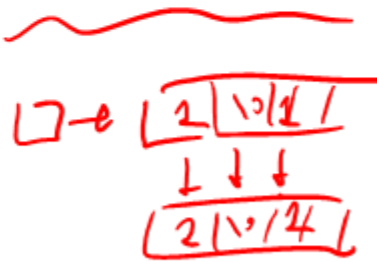
Pointer Example: strcpy()

```
char source_string[30] = "COPY ME!!!";
char dest_string[30] = "please don't copy over me :(";

strcpy(source_string, dest_string);
printf ("%s\n", dest_string );
```

```
feds (orig_string);
RPN_Eval (orig_string);
```

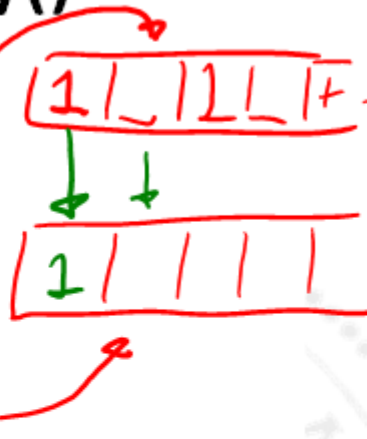



```
RPN_Eval (orig string) {
    char * dup_string = orig_string;
    strtok (dup_string);
    char dupstring[100];
    strcpy (orig_str, dup_str)
```



Pointer Example: strcpy()

```
void myStrcpy(char * source, char * dest)
{
    while (TRUE) {
        *dest++ = *source++;
        if (*source == '\0') return;
    }
}

void main(void){
    char source_string[30] = "COPY ME!!!";
    char dest_string[30] = "please don't copy over me :(";
    myStrcpy(source_string, dest_string);
    printf ("%s\n", dest_string );
```



char * dest = "blah blah";

vs

char source[30] = "";

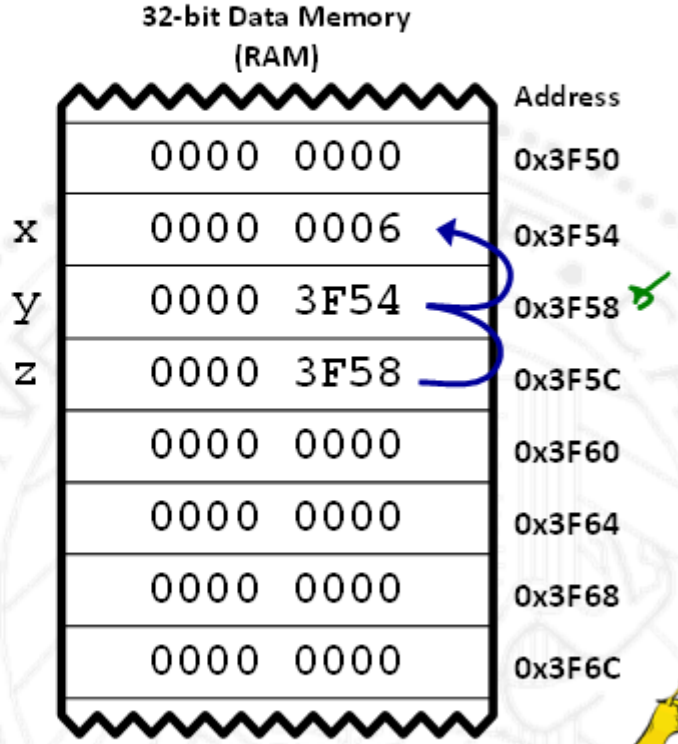
Pointers to Pointers

Example

```
{
  int x = 6;

  int *y = &x;

  int **z = &y;

  printf("%d\n", **z);
}
```

Output

```
6
```



```
char ** strlist = { "Hello", "world", "!" };
char * stringlist[3] =
```

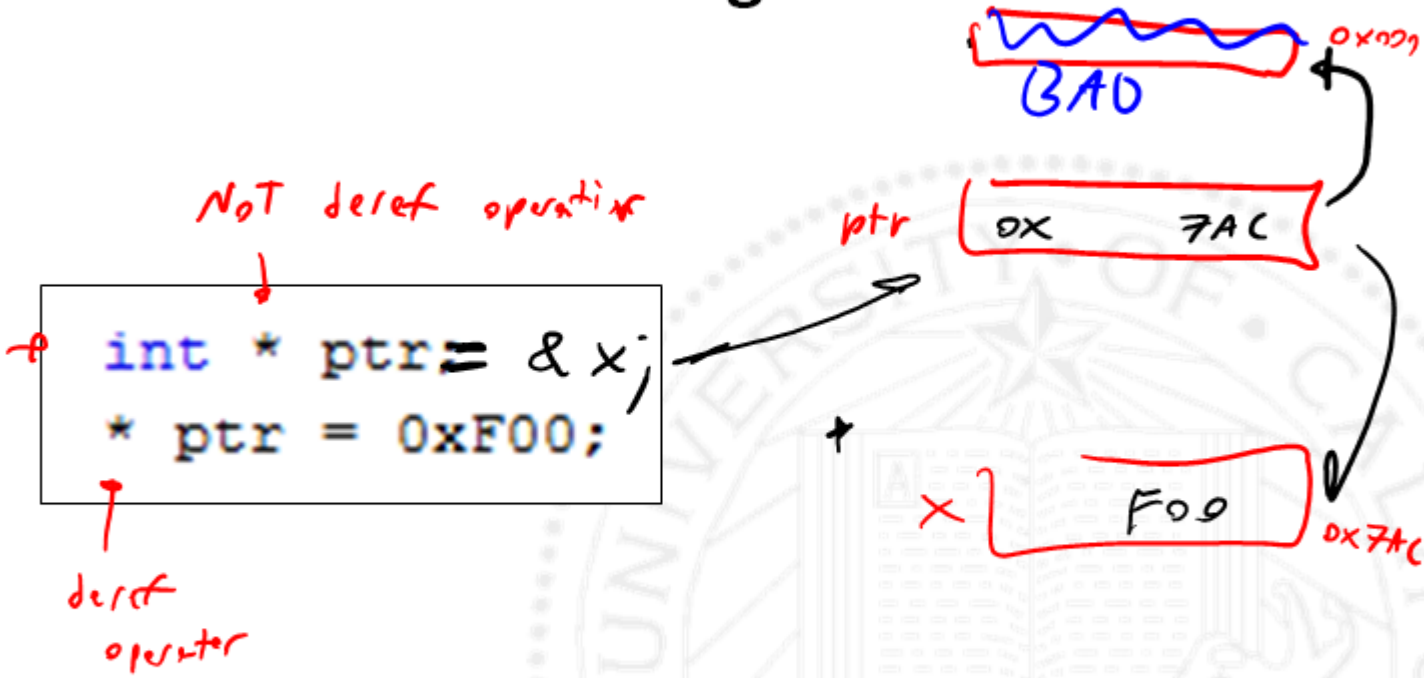
Pointers

Passing by reference, again

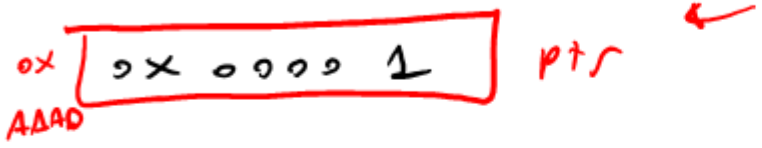
- If a pointer is passed to a function, the data it points to can be altered
- If a pointer-to-a-pointer is passed, the pointer it points to **and** the data that pointer points to can be altered

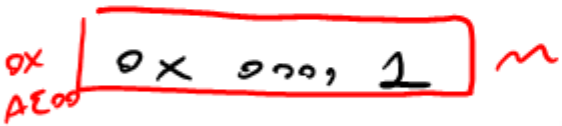
Common Pointer Errors

What's wrong with this?



Is anything wrong with this?






```
int * ptr;
int m = 30;
ptr = m;
```

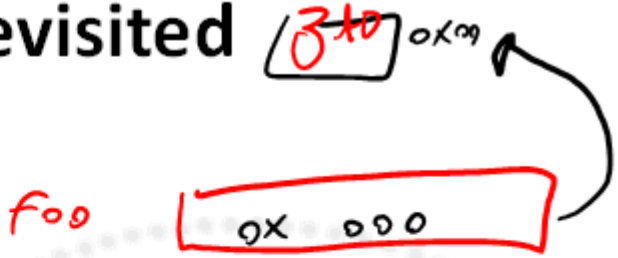
Max Lichtenstein

Something's definitely wrong with this, but what?

```
int getNextValue(double * nextValue){
.
.
}

main(){
    double myVal;
    getNextValue(&myVal);
```

Max Lichtenstein

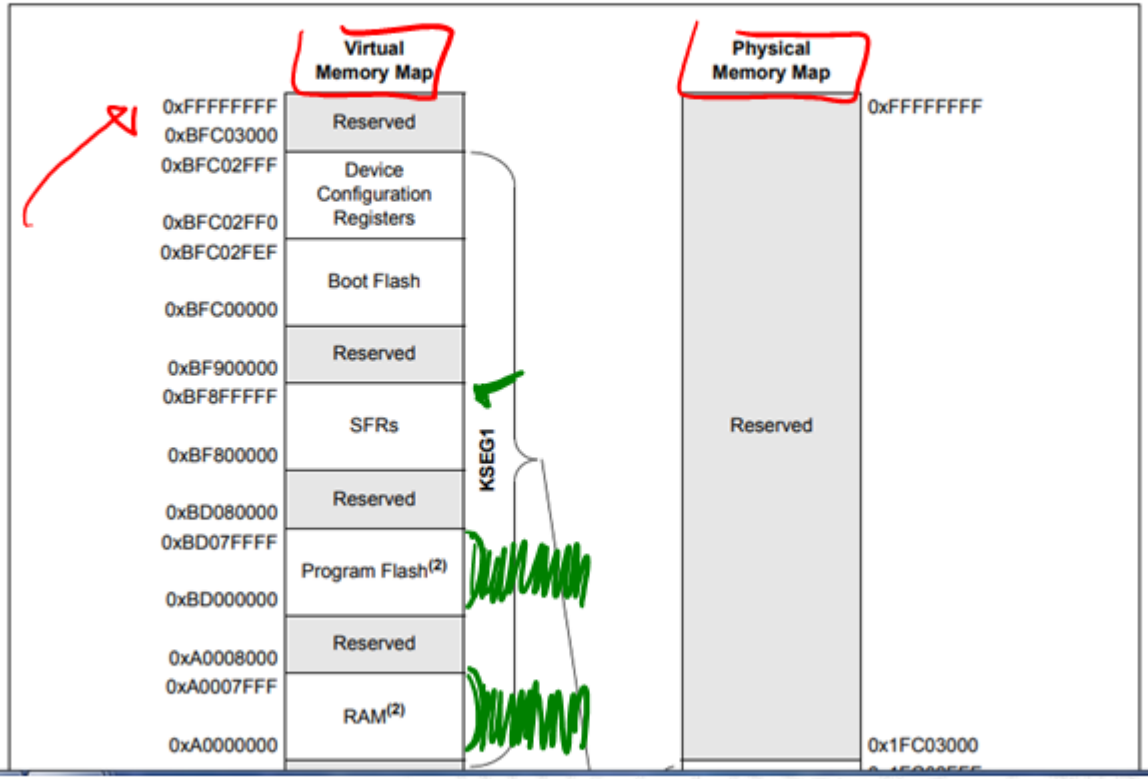
Piazza poll revisited

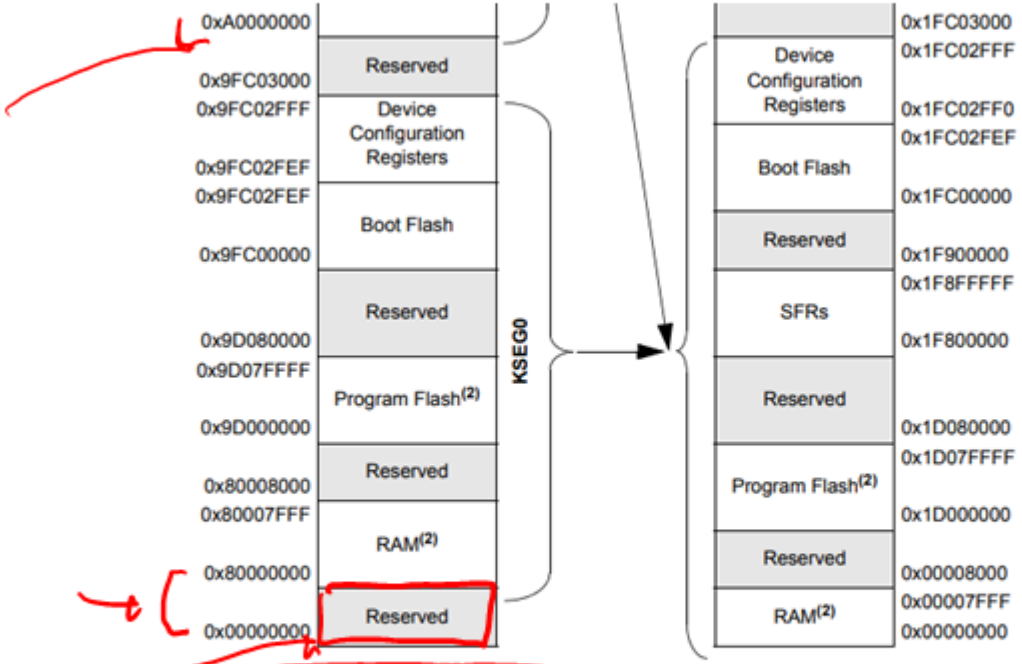



```
float * foo = NULL;
*foo = 77;
printf("foo = %x", foo);
```

Max Lichtenstein

FIGURE 4-6: MEMORY MAP ON RESET FOR PIC32MX340F512H, PIC32MX360F512L, PIC32MX440F512H AND PIC32MX460F512L DEVICES(1)





Note 1: Memory areas are not shown to scale.

2: The size of this memory region is programmable (see **Section 3. "Memory Organization"** (DS61115)) and can be changed by initialization code provided by end-user development tools (refer to the specific development tool documentation for information).

Are these legal?

```
int * red;
int white;
int & blue;
```

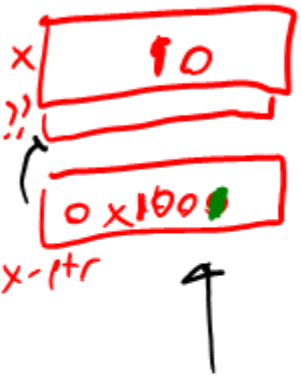
```
void foo( int &bar);
```

Pointer puzzles

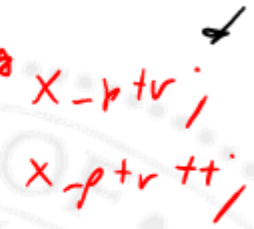
What does it do?

```
void doubleEach(int * x, int * y, int * z){
    *x = (*x) * 2;
    *y = (*y) * 2;
    *z = (*z) * 2;
}

int main(void){

    int X = 3, Y = 4;
    int *Z = 10;
    doubleEach(&X, &Y, &Z);

    printf("xyz = %d, %d, %d\n", X, Y, Z);
```

Piazza Poll: What does it output?




```
int x = 10;
int *x_ptr = &x;

*x_ptr++;

printf("x = %d\n", x);
```



A: x = 10

B: x = 11

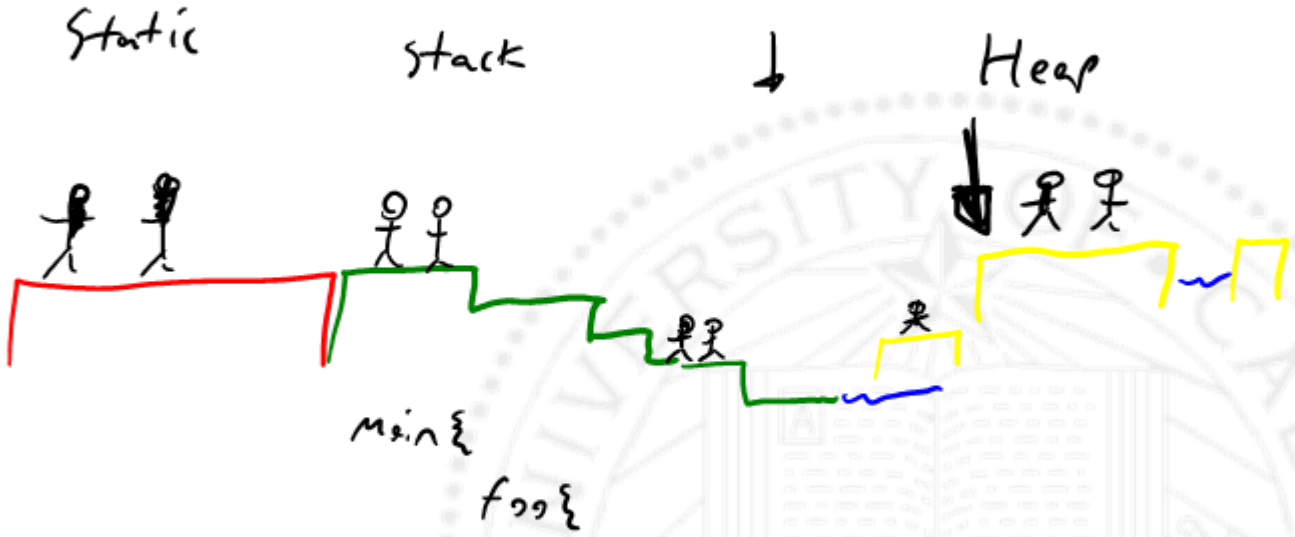
C: x = <some garbage>

D: runtime exception

Max Lichtenstein

BREAK

Dynamic memory



Dynamic Memory

malloc()

Syntax

```
void *malloc(size_t size);
```

- Request memory of **`size`** bytes
  - Usually returned by **`sizeof`** operator
- Returns valid pointer or NULL

Example

```
typedef struct {
  float re;
  float im;
} Complex;

Complex *x = malloc(sizeof(Complex));
```

Dynamic Memory

The Heap

Example

```
typedef struct {
  float re;
  float im;
} Complex;

Complex *x = malloc(sizeof(Complex));
```

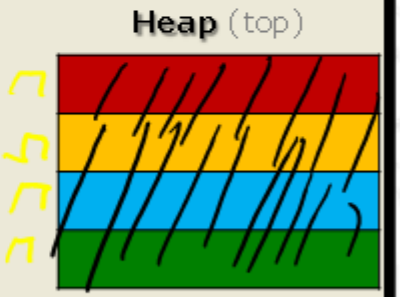
Dynamic Memory

The Heap

Example

```
typedef struct {
  float re;
  float im;
} Complex;

Complex *x = malloc(sizeof(Complex));
Complex *x = malloc(sizeof(Complex));
Complex *x = malloc(sizeof(Complex));
Complex *x = malloc(sizeof(Complex));
Complex *x = malloc(sizeof(Complex));
```

NULL



Dynamic Memory

NULL pointers

Example

```
typedef struct {
  float re;
  float im;
} Complex;

Complex *x = malloc(sizeof(Complex));
if (x) {
  x->re = 0.0;
}
x->im = 0.0;
printf("Complex{re:%f im:%f}\n", x->re, x->im);
```

Dynamic Memory

free()

Syntax

```
void free(void *ptr);
```

- Frees memory pointed to by `ptr`
  - **Must** have been returned by `malloc()`

Example

```
typedef struct {
  float re;
  float im;
} Complex;

Complex *x = malloc(sizeof(Complex));
free(x);
```

malloc() and free() example

Lab 5

- Doubly Linked Lists
- Allocation
- String Comparison
- Sorting
- Algorithm Analysis

Linked Lists

Doubly Linked Lists

Lab 5 in C

```
typedef struct ListItem {
    struct ListItem *previousItem;
    struct ListItem *nextItem;
    char *data;
} ListItem;
```

Lab 5 in C

```
typedef struct ListItem {
    struct ListItem *previousItem;
    struct ListItem *nextItem;
    char *data;
} ListItem;
```

```
ListItem *LinkedListNew(char *data);
```

Max Lichtenstein