Week 4

This Week

- Style
- Merge Sort
- Pointers
- Dynamic Memory Allocation
  - Malloc
  - Free

Style

Good Style:

- Consistent Indentation
- Comments Appropriately Detailed
- Self-Explanatory Variable Names

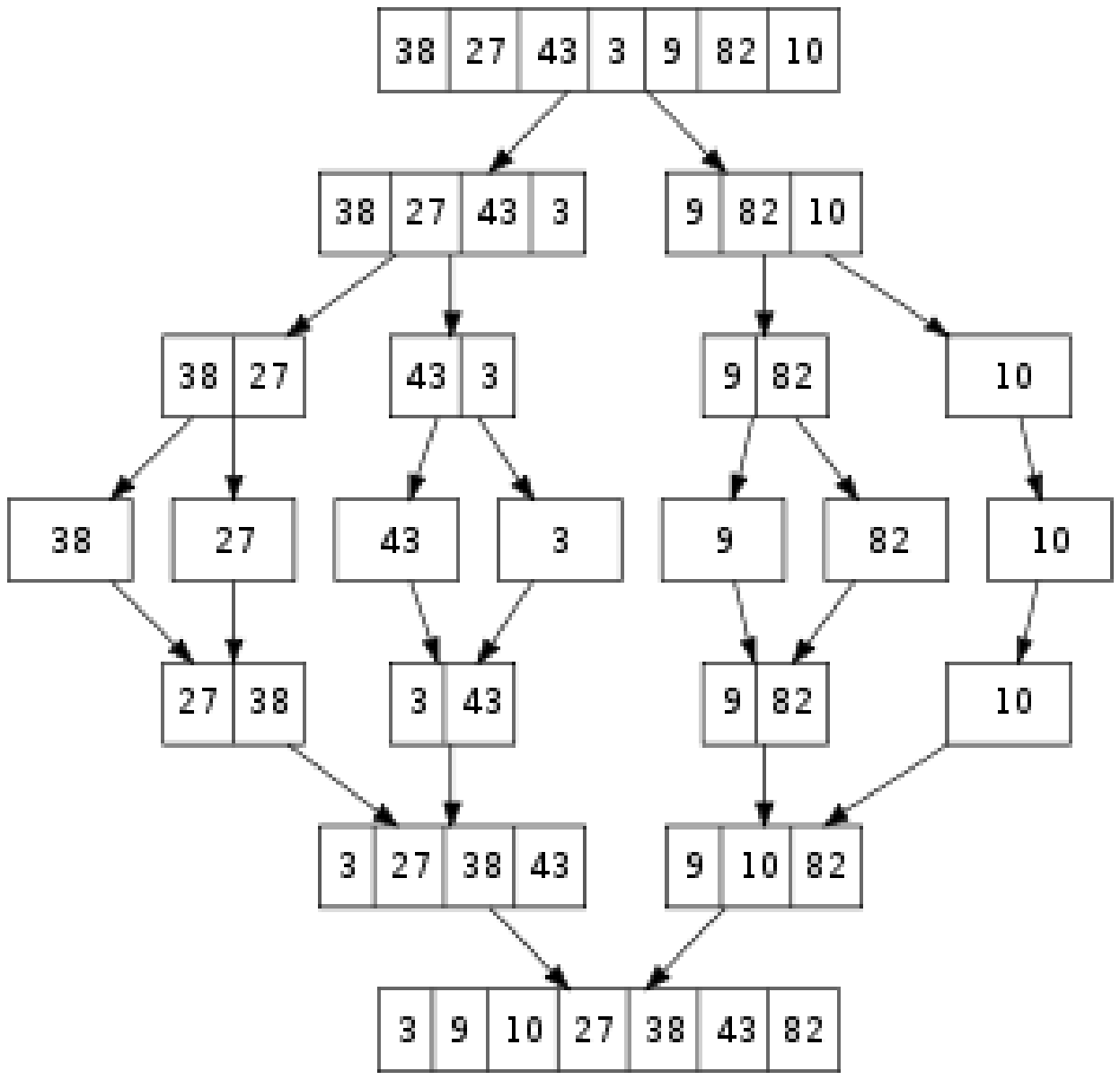
Bad Style:

- Single-letter or cryptic variable names
- Lack of comments; paragraph comments

Merge Sort

Sorting Algorithm which recursively breaks lists into halves, then combines the ‘sorted’ base cases into larger sorted lists.

Merge Sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Merge Sort

What sort of asymptotic runtime does this algorithm have?

O(n log n)

Pointers

- Data stored in memory has both a value and a *location*.
- Pointers contain the **memory address** of some piece of data.

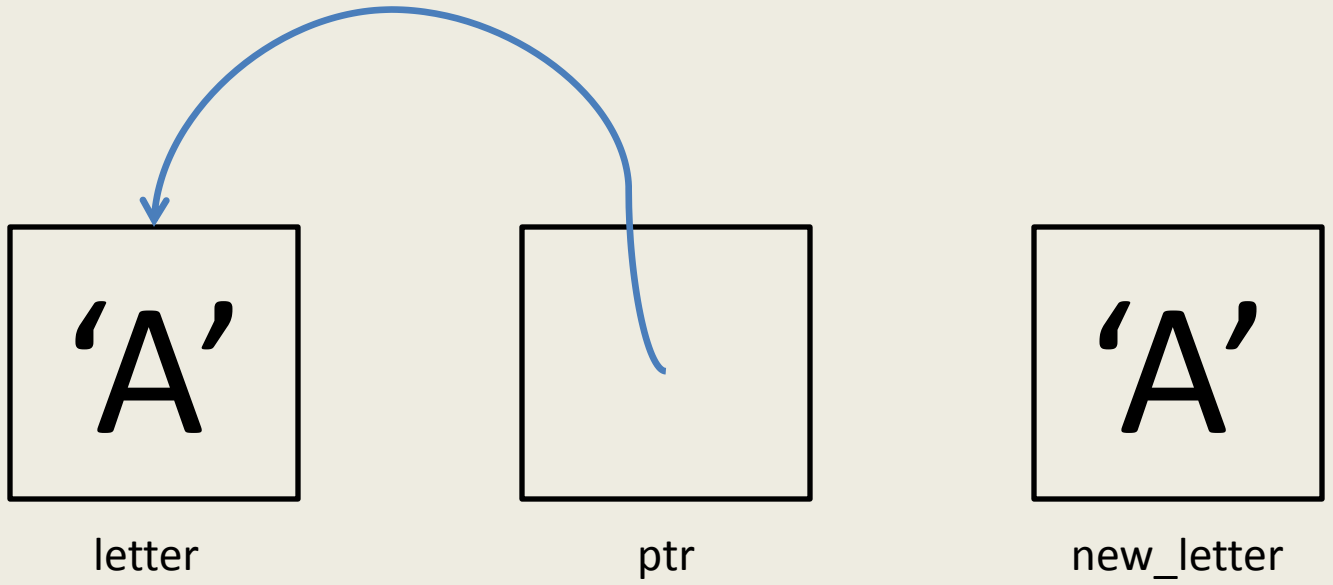
- <type>* pointer contains address to a piece of data of that type

Pointers

- & - Address operator; gets the address of a variable.
- * - Dereference operator; gets value found at the memory address to which a pointer points.

Pointers

```
char letter = ‘A’;
char* ptr = &letter; // get address of variable
char new_letter = *ptr; // dereference to get value
```



Pointers

Pointer arithmetic: adding *n* to a pointer shifts the pointer over by

*n**sizeof(<type of the pointer>) bytes

Pointers

```
int x;
int* y = &x;
y += 1;
```

If the address of x is 0x04, what is y now?

Pointers

```
char x;
char* y = &x;
y += 1;
```

If the address of x is 0x04, what is y now?

What about now?

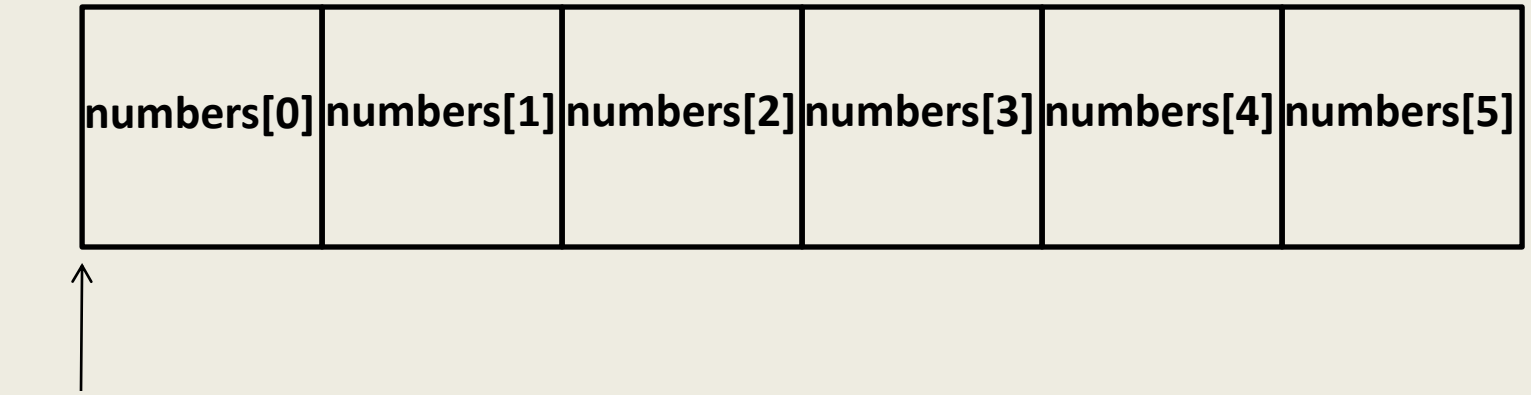
Pointer Practice

```
int a = 3, b = 4, c = 5;
int *pa = &a, *pb = &b, *pc = &c;
```

	a	b	c	pa	pb	pc
a = b * c;	20	4	5	&a	&b	&c
a *= c;	100	4	5	&a	&b	&c
b = *pa;	100	100	5	&a	&b	&c
pc = pa;	100	100	5	&a	&b	&a
*pb = b * c;	100	500	5	&a	&b	&a
c = (*pa) * (*pc);	100	500	10000	&a	&b	&a
*pc = a * (*pb);	50000	500	10000	&a	&b	&a

Arrays are Pointers!

```
int numbers[6];
```



Arrays are Pointers!

array[i] == *(array + i)

Dynamic Memory Allocation

- malloc(int num_bytes)
  - Reserves a region of memory on the heap of size num_bytes.
  - Returns the address of this region of memory.
- free(void* pointer)
  - Takes as an argument the address of some region of memory reserved by malloc.
  - Frees up the memory which was reserved.

Dynamic Memory Allocation

When using malloc:

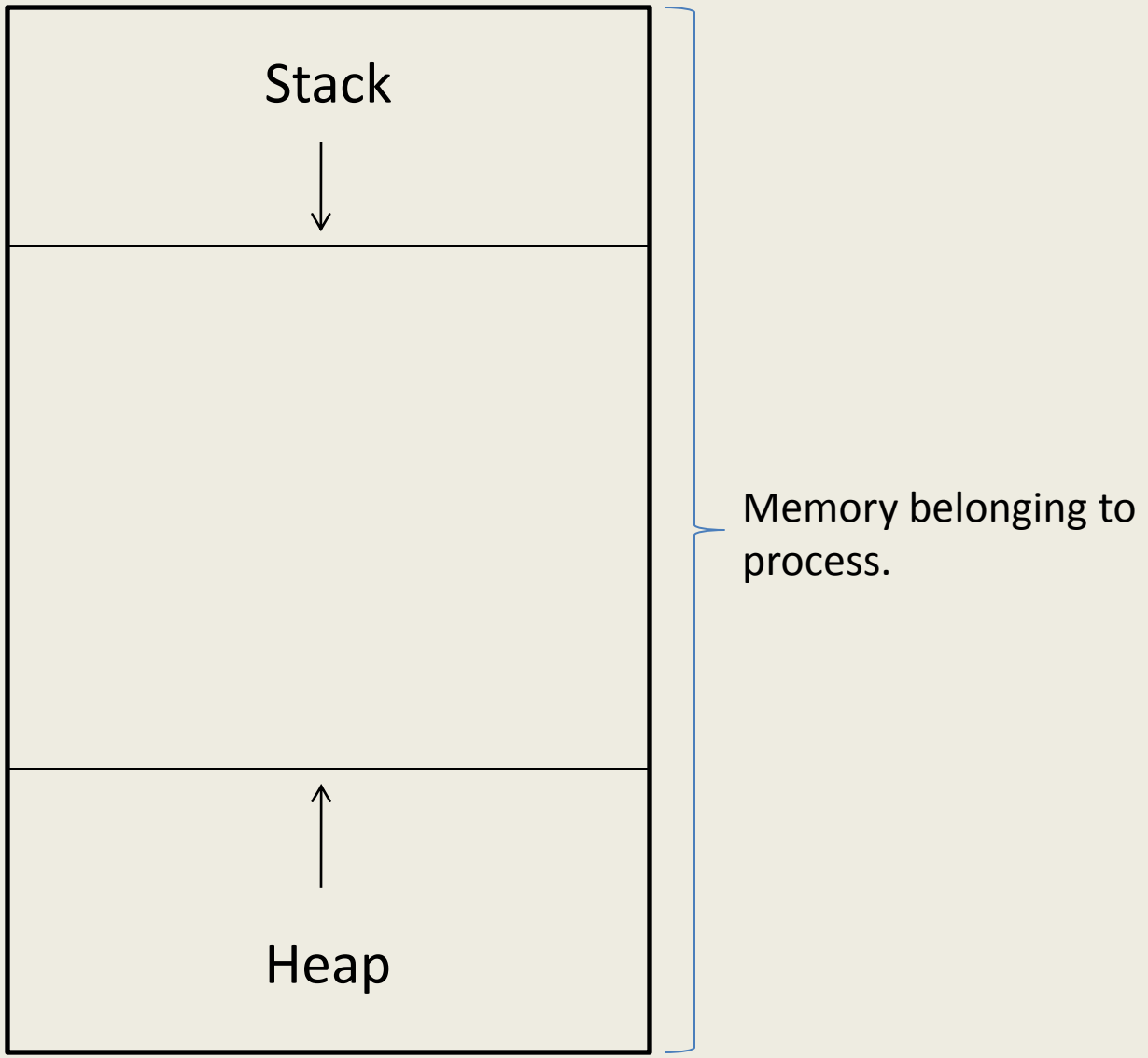
Always check whether it returns ‘null’.

Free allocated memory *exactly once*.

```
char* x = malloc(sizeof(char)*10);
free(x);
```

Stack vs. Heap

- Contains local variables.
- Function calls create new ‘frames’ on the stack.



- Contains global variables.
- Dynamically allocated memory reserved on heap.