CSE 220 – C Programming

malloc, calloc, realloc

Outline

- Arrays and Pointers
- Memory allocation
- Memory deallocation

Is this code legal?

```
int a[10] = {1, 2, 3};
for (int * p = &a[0]; p < &a[10]; p++) {
      printf("%d\n", *p);
}
```

Legal

Illegal (Array Initialization)

Illegal (Pointer Arithmetic)

I don't know

Using pointers to arrays

- Can traverse array by incrementing a pointer to it

```
int a[N], *p, sumAll = 0, sumPartial = 0;
for (p=&a[0]; p<&a[N]; p++)  //add every element
    SumAll += *p;
for (p=&a[0]; p<&a[N]; p=p+2)  //add every other element
    SumPartial += *p;
```

Using array name as pointer

- The name of an array is a pointer to the first element of the array

```
int a[N];
*a = 8;          //Puts 8 in a[0]
*(a+1) = 9;      //Puts 9 in a[1]
for (p=a; p<a+N; p++)
   sumAll += *p;
```

Is this code legal?

```
int array[10] = {1, 2, 3};
while (*array != 0) {
      array++;
}
```

Legal

Illegal (While Condition)

Illegal (Increment Operation)

I don't know

Using array name as pointer

- It is not possible to assign a new value to the array name

```
int a[N];
while (*a != 0)
   a++;        //WRONG!
```

- Instead, use a temp pointer that moves through the array

```
int *p = a;
while (*p != 0)
   p++;
```

Dynamic Storage Allocation

- Fixed size data structures have the same number of elements from compilation time for the whole structure lifetime:

```
int a[100];

//Will always have 100 elements
```

- Dynamic storage allocation: the ability to allocate storage during program execution
  - Design data that grows and shrinks as needed
  - Normally used for strings and arrays

Memory allocation functions

- malloc: allocates a block of memory without initializing
- calloc: allocates a block of memory and clears it
- realloc: resizes a previously allocated block of memory
- These functions are declared in the <stdlib.h>
- Take as input the number of bytes to allocate

Memory allocation functions

```
char *pc = malloc(N+1);
int *pi = malloc (400);
```

- Does malloc return int * or char *?
  - malloc does not know the type of data that will be stored in the block of memory so it returns a generic pointer: void *

```
char *pc = (char *) malloc(N+1);
int *pi = (int *) malloc (400);
```

- If allocation fails, malloc returns null pointer: NULL

Dynamically Allocated Strings

Write a function (named "concat") that takes two strings s1 and s2 and returns a third string obtained by concatenating s1 and s2

Allocate enough memory to hold s1 and s2:
size of s1 + size of s2 + 1 (to fit \0)
Write s1 in s3
Write s2 in s3 right after s1
Add the termination character \0

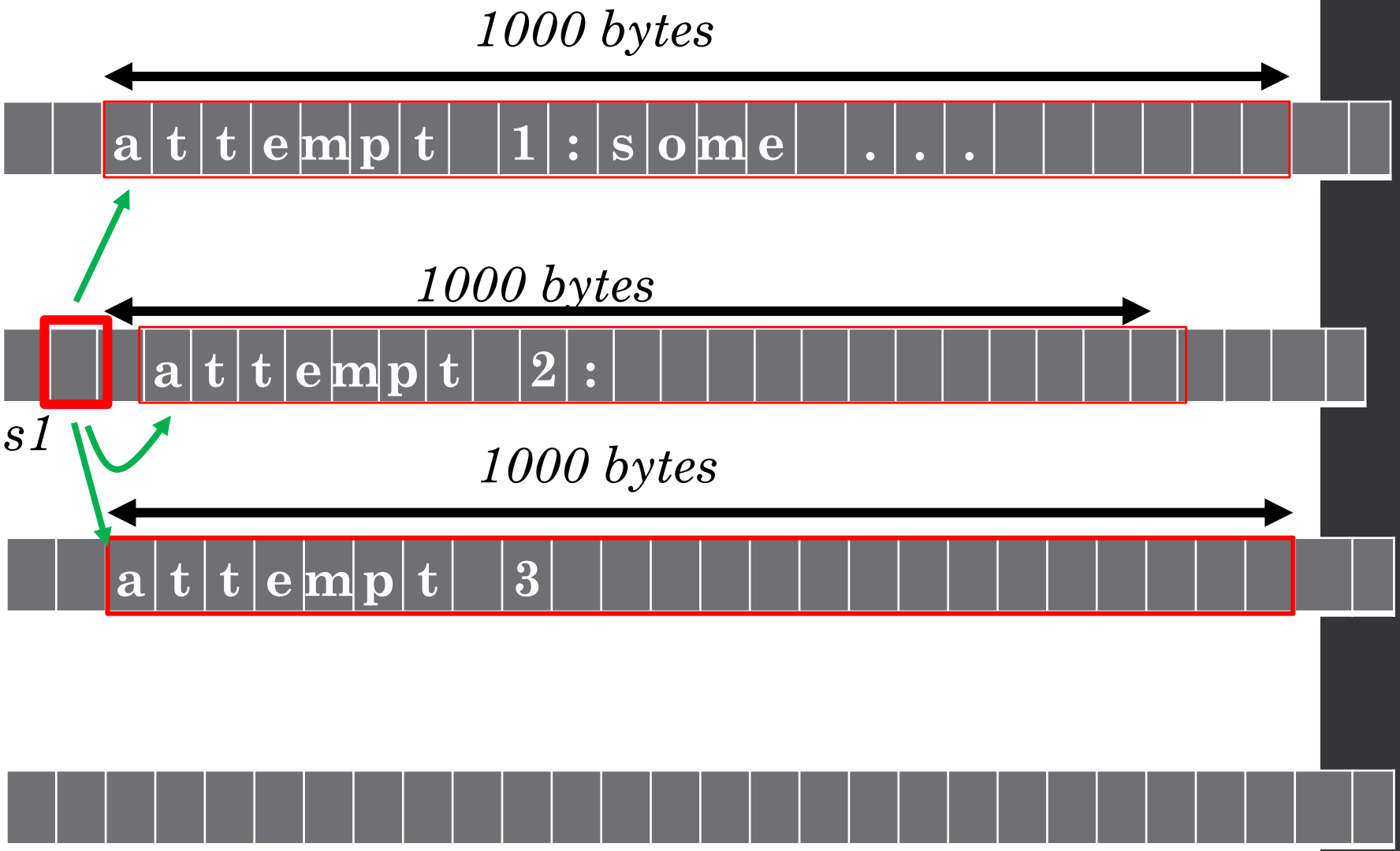
Dynamically Allocated Strings

```
char * concat(char *s1, char *s2) {
    char * result;
    //Points to nothing in particular
    result = malloc(strlen(s1) + strlen(s2) + 1);
    /* If malloc succeeds, result points to new
    allocated memory */
    if (result == NULL) {
    //Check if allocation succeeded
        printf("Error: could not allocate memory\n");
    }
    strcpy(result, s1);
    strcat(result, s2);
    return result;
 }
```

Deallocating memory

- Memory allocated with malloc lives for the lifetime of the program

```
char *s1 = (char *) malloc(1000);
strcpy(s1, “attempt 1: some long text”);
s1 = (char *) malloc(1000);
strcpy(s1, “attempt 2: just another string”);
s1 = (char *) malloc(1000);
strcpy(s1, “attempt 3: one last time……”);
```

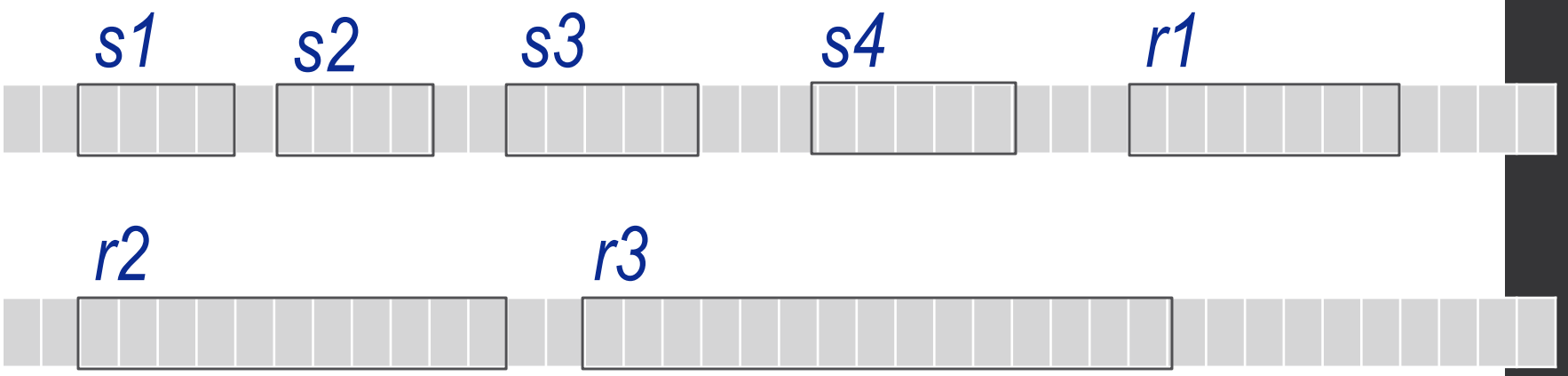

1000 bytes
attempt 1: some ...
1000 bytes
attempt 2:
s1
1000 bytes
attempt 3

Deallocating memory

- Memory allocated with malloc lives for the lifetime of the program
- If no longer needed, make sure to free the memory used otherwise program might run out of memory.

```
char *s1 = “Electric “, *s2 = “current “,
char *s3 = “is measured ”, char *s4 = “in amperes (amps)”;
char *r1 = concat(s1, s2);
char *r2 = concat(r1, s3);
char *r3 = concat(r2, s4);
```

A look at memory



Make sure to free memory that is not needed:

```
free(r1);

free(r2);
```

Why is it important to release memory that is no longer needed?

Computers have a finite amount of memory, so a program that uses too much may crash.

Programs that use more memory often run slower.

If you love it, set it free.

I don't know

Arrays of strings

```
char tmp[20];
char * winningCars[5];  //Declare array of pointers
//Each pointer still points to nothing in particular
for (int i=0; i<5; i++) {
    scanf(“%s”, tmp);
    //Allocate memory for each pointer
    winningCars[i] = malloc(strlen(tmp) + 1);
    strcpy(winningCars[i], tmp);
}
```

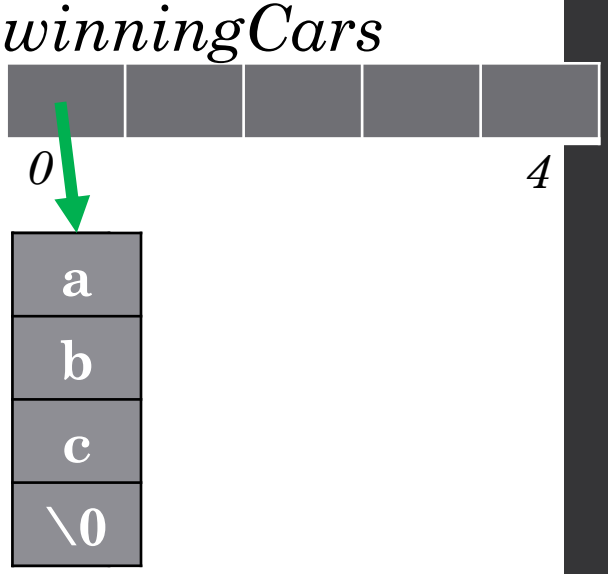
Arrays of strings


```
char tmp[20];
char * winningCars[5];
```


```
scanf("%s", tmp);
winningCars[0] = malloc(strlen(tmp) + 1);
strcpy(winningCars[0], tmp);
```

Arrays of strings

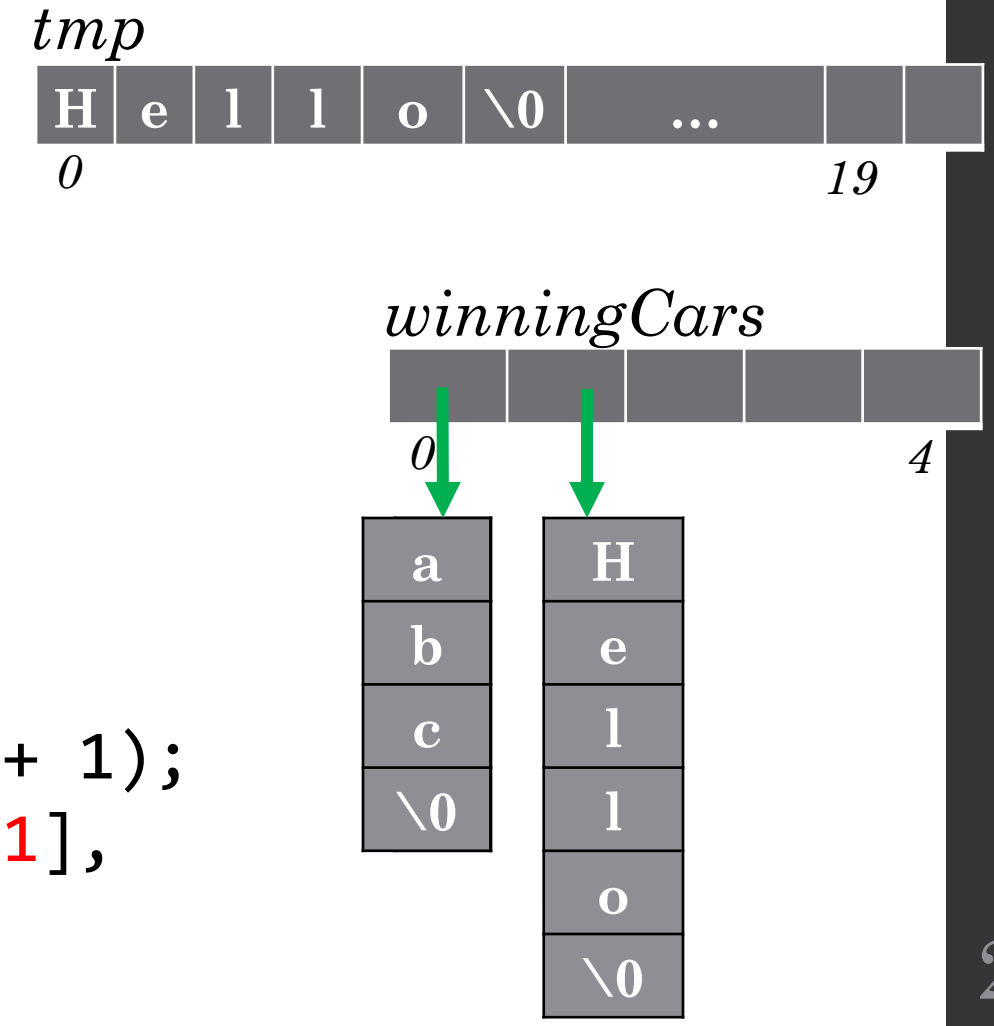
```
char tmp[20];
char * winningCars[5];
```




```
scanf("%s", tmp);
winningCars[0] =
malloc(strlen(tmp) + 1);
strcpy(winningCars[0], tmp);
```

Arrays of strings

```
char tmp[20];
char *
winningCars[5];
```




```
scanf("%s", tmp);
winningCars[1] =
malloc(strlen(tmp) + 1);
strcpy(winningCars[1],
tmp);
```

Dynamically Allocated Arrays

```
int *a = malloc(400);
   //Allocates 400 bytes
```

- How many elements in a?
  - If int consists of 4 bytes, then a has 100 elements

```
int *a = malloc (N * sizeof(int));
   //N elements
```

- Once memory is allocated, treat *a* as any other array:

```
a[1] = 5;
*(a + 2) = 7;
```

Dynamically Allocated Arrays

- `int *a = malloc (N * sizeof(int)); //N elements`
- `int *b = calloc (N, sizeof(int));  //N elements`
- A has enough memory for n integers
- B has enough memory for n integers.
- All elements of B are initialized to zeros.

Why not always use calloc (instead of malloc)?

It has a more silly sounding name.

It is slower (memory needs to be cleared first).

It can hold fewer variables in the same space.

I don't know

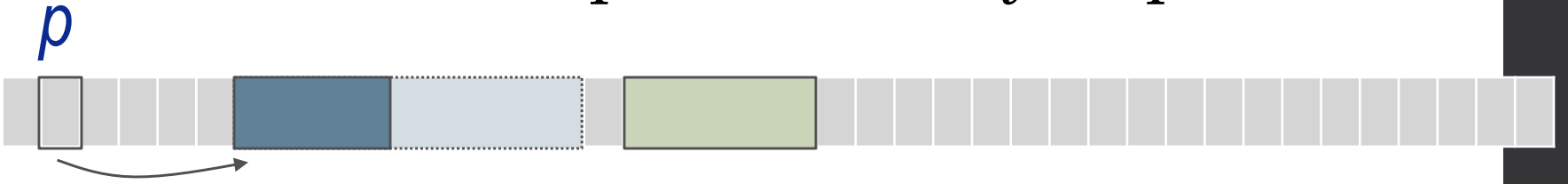
Dynamically Resize Arrays

- If previously allocated memory is too small or too big, can resize with *realloc*
- When calling *realloc*, the pointer given must be to memory allocated using malloc, calloc or realloc

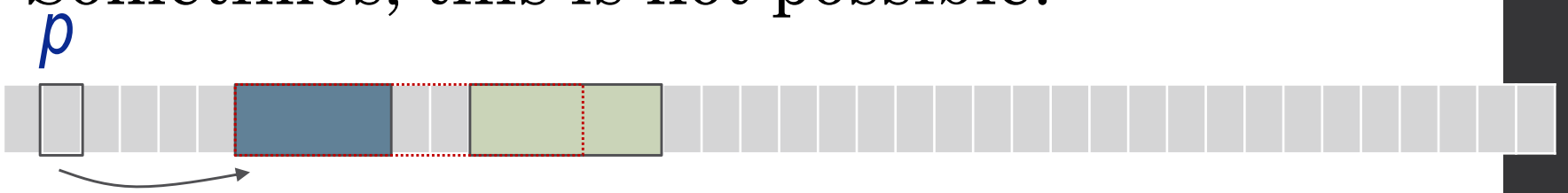
```
char *str =  malloc(N + 1);
….
str = realloc (str, 2*N + 1);
```

Dynamically Resize Arrays

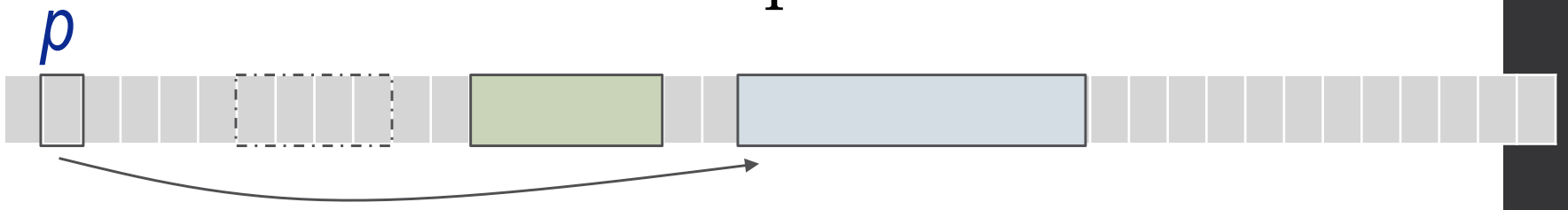
realloc tries to expand memory in place:



Sometimes, this is not possible:

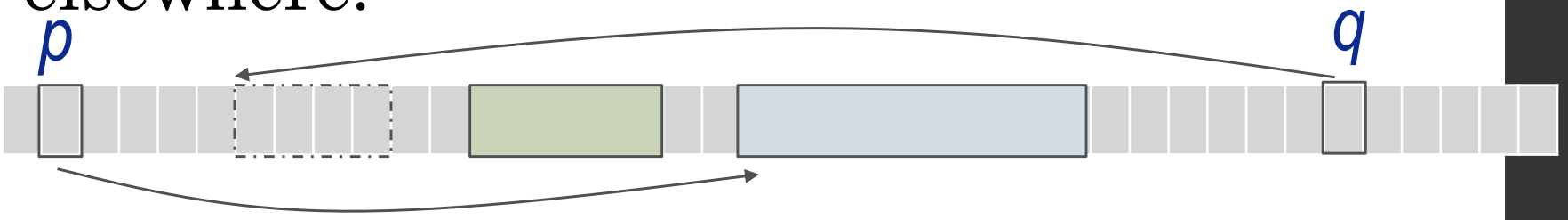


Data is moved then expanded:



Dynamically Resize Arrays

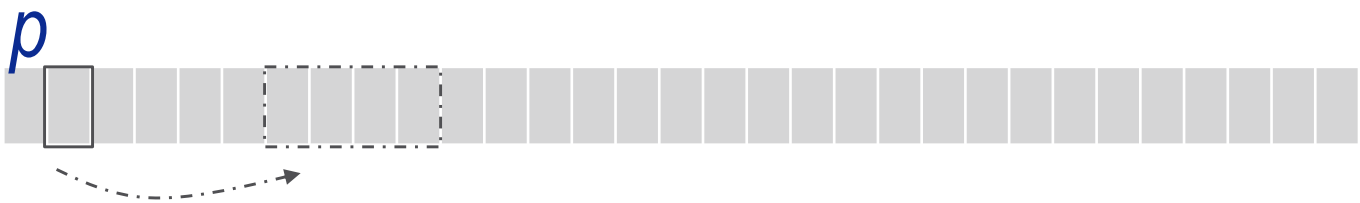
There could be other pointers pointing to the old location. Make sure to update all the pointers since the block could have moved elsewhere.




```
//Update q to point to the same location as p
q = p;
```

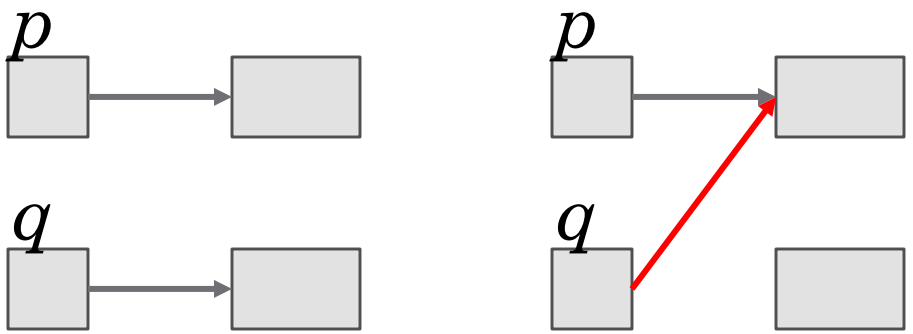
Deallocating memory

- free(ptr): deallocates the block of memory pointed to by ptr



- May lose track of memory blocks:

```
p = malloc(…);
q = malloc(…);
q = p;
```



Deallocating memory

- Memory without a pointer to it is called garbage
- A program that leaves garbage behind has a memory leak
- C does not have automatic garbage collection. It is the programmer's job
- Pointer freed becomes a dangling pointer: does not have memory associated with it. It cannot be reused without being allocated some memory.
- If several pointers point to one location and one pointer is freed, all the pointers become dangling.