malloc()

STYLE TIP

- General usage: `var = (type*) malloc(size in bytes)`
- `malloc` allocates a block of memory of the requested size
  - Returns a pointer to the first byte of that memory
    - And returns `NULL` if the memory allocation failed! // Check this!
  - You should assume that the memory initially contains garbage
  - You'll typically use `sizeof` to calculate the size you need

```
// allocate a 10-float array
float* arr = (float*) malloc(10*sizeof(float));
if (arr == NULL) {
  return errcode;
}
...   // do stuff with arr
```


free()

DEBUG TIP

- Usage: `free(pointer);`
- Deallocates the memory pointed-to by the pointer
  - Pointer *must* point to the first byte of heap-allocated memory (*i.e.* something previously returned by `malloc` or `calloc`)
  - Freed memory becomes eligible for future allocation
  - `free(NULL);` does nothing.
  - The bits in the pointer are *not changed* by calling free
    - Defensive programming: can set pointer to `NULL` after freeing it

```
float* arr = (float*) malloc(10*sizeof(float));
if (arr == NULL)
  return errcode;
...            // do stuff with arr
free(arr);
arr = NULL;    // OPTIONAL
```


Poll Everywhere

pollev.com/cse33320su

- Which line below is first *guaranteed* to cause an error?

A. Line 1
B. Line 4
C. Line 6
D. Line 7
E. We're lost…

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

int main(int argc, char** argv) {
  int a[2];
  int* b = malloc(2*sizeof(int));
  int* c;

1 a[2] = 5;
2 b[0] += 2;
3 c = b+3;
4 free(&(a[0]));
5 free(b);
6 free(b);
7 b[0] = 5;

  return 0;
}
```


typedef

- Generic format: `typedef type name;`
- Allows you to define new data type *names/synonyms*
  - Both `type` and `name` are usable and refer to the same type
  - Be careful with pointers – `*` before `name` is part of `type`!

```
// make "superlong" a synonym for "unsigned long long"
typedef unsigned long long superlong;

// make "str" a synonym for "char*"
typedef char *str;

// make "Point" a synonym for "struct point_st { ... }"
// make "PointPtr" a synonym for "struct point_st*"
typedef struct point_st {
  superlong x;
  superlong y;
} Point, *PointPtr;  // similar syntax to "int n, *p;"

Point origin = {0, 0};
```