CSC209 Lecture 3: Dynamic Memory Allocation

David Liu, Department of Computer Science, University of Toronto

*Navigation tip for web slides: press **?** to see keyboard navigation controls.*

Announcements

- Assignment 1 due date extended to **Monday, January 30 6:30pm**

Story so far:

- Local variables are stored in **stack frames**
- Stack frames are automatically allocated when functions are called
- Stack frames are automatically deallocated when functions return
- Pro: don't have to worry about "cleaning up" local variables after function ends
- Con: objects defined through local variables can't be accessed after function ends

```
int *get_nums() {
    int arr[3] = {10, 20, 30};
    return arr;
}
```

```c
#include <stdlib.h>
void *malloc(size_t size);
```

- `malloc` allocates `size` bytes of memory on the heap
  - guaranteed to be disjoint from other allocated memory
- `malloc` returns a pointer to the start of the allocated memory
- Pro: can allocate memory in one function that can be accessed after function ends
- Con: must manually deallocate memory after it is no longer used

```c
#include <stdlib.h>
void free(void *ptr);
```

- when `ptr` is a pointer previously returned by `malloc`, `free` deallocates the memory that was previously allocated

Worksheet: malloc_basics.pdf

Worksheet: stack_vs_heap.pdf

Pitfalls with dynamic memory

A **memory leak** is when some heap-allocated memory is never freed.

A **use-after-free** error is when code dereferences a pointer that has been freed. (Such pointers are called dangling pointers.)

A **double free** error is when `free` is called twice on the same pointer.