Stack vs Heap

Adapted from materials by Dr. Carrier

Monika Borys- Unsplash

Mandy Jouan - Flickr

The what and what??

The what and what??

- Two areas of memory

The what and what??

- Two areas of memory
- The stack is a stack data structure

The what and what??

- Two areas of memory
- The stack is a stack data structure
- The heap is not necessarily a heap 🫠

The what and what??

- Two areas of memory
- The stack is a stack data structure
- The heap is not necessarily a heap 🫠
- Variables are in one or the other
  - Which one depends on how variable is declared!

The what and what??

- Two areas of memory
- The stack is a stack data structure
- The heap is not necessarily a heap 🫠
- Variables are in one or the other
  - Which one depends on how variable is declared!
- So far, we’ve only dealt with the stack

The stack

The stack

- New block (scope) for each function called

The stack

- New block (scope) for each function called
  - Function local variables are stored in block

The stack

- New block (scope) for each function called
  - Function local variables are stored in block
  - When function returns, that block is freed

The stack

- New block (scope) for each function called
  - Function local variables are stored in block
  - When function returns, that block is freed
    - Those variables are gone!

The heap

The heap

- Dynamically allocated memory

The heap

- Dynamically allocated memory
  - Not tied to a particular functions scope

The heap

- Dynamically allocated memory
  - Not tied to a particular functions scope
  - Can be allocated and freed at any time

The heap

- Dynamically allocated memory
    - Not tied to a particular functions scope
    - Can be allocated and freed at any time
- We must free any memory we allocate

The heap

- Dynamically allocated memory
  - Not tied to a particular functions scope
  - Can be allocated and freed at any time
- We must free any memory we allocate
  - What happens if not?

The heap

- Dynamically allocated memory
  - Not tied to a particular functions scope
  - Can be allocated and freed at any time
- We must free any memory we allocate
  - What happens if not?
    - Memory leaks!

The heap

- Dynamically allocated memory
  - Not tied to a particular functions scope
  - Can be allocated and freed at any time
- We must free any memory we allocate
  - What happens if not?
    - Memory leaks!
      - You can lose access (pointer) to allocated memory
      - Thus your program *can't* free it
      - Usually cleaned up by OS when program exits

Why have both?

Why have both?

- Stack allocates / deallocates automatically

Why have both?

- Stack allocates / deallocates automatically
    - Often limited in size

Why have both?

- Stack allocates / deallocates automatically
    - Often limited in size
    - All allocations on stack are static (sizes are set at compile time)

Why have both?

- Stack allocates / deallocates automatically
  - Often limited in size
  - All allocations on stack are static (sizes are set at compile time)
- Heap allows flexibility, but require more thought

Why have both?

- Stack allocates / deallocates automatically
  - Often limited in size
  - All allocations on stack are static (sizes are set at compile time)
- Heap allows flexibility, but require more thought
  - Larger arrays

Why have both?

- Stack allocates / deallocates automatically
  - Often limited in size
  - All allocations on stack are static (sizes are set at compile time)
- Heap allows flexibility, but require more thought
  - Larger arrays
  - Arrays that persist after a function returns

Why have both?

- Stack allocates / deallocates automatically
  - Often limited in size
  - All allocations on stack are static (sizes are set at compile time)
- Heap allows flexibility, but require more thought
  - Larger arrays
  - Arrays that persist after a function returns
  - Amount of memory is unknown at compile time

Why have both?

- Stack allocates / deallocates automatically
  - Often limited in size
  - All allocations on stack are static (sizes are set at compile time)
- Heap allows flexibility, but require more thought
  - Larger arrays
  - Arrays that persist after a function returns
  - Amount of memory is unknown at compile time
    - *Technically possible in stack via VLAs
      - But we’re ignoring this ;^)

Memory allocation

Memory allocation

- Stack

Memory allocation

- Stack
    - Static (compile time) memory allocation
        - Examples: `int x; char arr[20];`

Memory allocation

- Stack
    - Static (compile time) memory allocation
        - Examples: `int x; char arr[20];`
- Heap
    - Dynamic (runtime) memory allocation

Memory allocation

- Stack
  - Static (compile time) memory allocation
    - Examples: `int x; char arr[20];`
- Heap
  - Dynamic (runtime) memory allocation
    - Examples: `malloc(); realloc();` etc.
    - This is our next lecture :^)