# Lecture 21

## Memory management with variable sized blocks

*CS 241: Foundations of Sequential Programs*
Winter 2018

Troy Vasiga et al
University of Waterloo

# Variable size blocks

Unlike fixed-sized blocks, requests are not just for a block of memory (i.e., `int alloc()`), but rather, for a block of memory that is (at least) a given size (i.e., `int alloc(size)`.

Life only gets more difficult.

Suppose our arena has size 100, and we have requests:

```
A = alloc(10)
B = alloc(20)
C = alloc(10)
D = alloc(10)
E = alloc(10)
free(B)
free(D)
```

# Issues that need addressing

Issues:

- Allocate:
- Reclaim:
- Initialization:

# Fragmentation

Internal Fragmentation:

External Fragmentation:

# Allocation issues

Searching the “free list”

# Allocation strategies: Description

- First-fit:
- Best-fit:
- Worst-fit

# Allocation strategies: Example

- First-fit:
- Best-fit:
- Worst-fit

# Allocation strategies: Implementation

- First-fit:
- Best-fit:
- Worst-fit

# What a block should look like

Extra information stored with the block

# Doing more with reclamation

What if we reclaim two blocks which end up being side by side?

# Four cases of reclamation

# A final word about reclamation

# Buddy System: Initialize

# Buddy System: Allocate

# Buddy System: Delete

# Final words about the buddy system

# Knuth's 50% rule

**Theorem:** At equilibrium, the number of holes tends to be approximately half the number of allocated areas.

**Corollary:** If there is as much free space as allocated space in the arena, the average hole size is double the average allocated area.

Consequence of this for the first fit strategy:

# Solving A11P5

You have to implement the operations:

- `int malloc(int size)`
- `int free(int size)`

Note:

- you will need to make blocks with extra information
- you will need to manage some free list of blocks
- review the ideas outlined in this lecture

# Archaic C/C++

- C, C++ have explicit memory management
- `delete` is a promise from a programmer to not use memory again
- never believe a promise from a programmer:

```
#include <iostream>
using namespace std;
int main() {
  int* b = new int[3];
  *b = 7;
  *(b+1) = 17;
  *(b+2) = 345;
  delete [] b;
  cout << *(b+1) << endl;
}
```

- Java and lots of modern languages do not trust programmers: they have implicit memory management. Programmers cannot directly access memory locations in Java.