Applied Algorithms

Lecture 3: Allocation and Efficiency

Today

- Discuss assignment submission (and this assignment)
- Finish up C
- Talk in more detail about efficiency
- (Maybe) start space-efficient edit distance

Pros and cons of running a local setup

Pros:

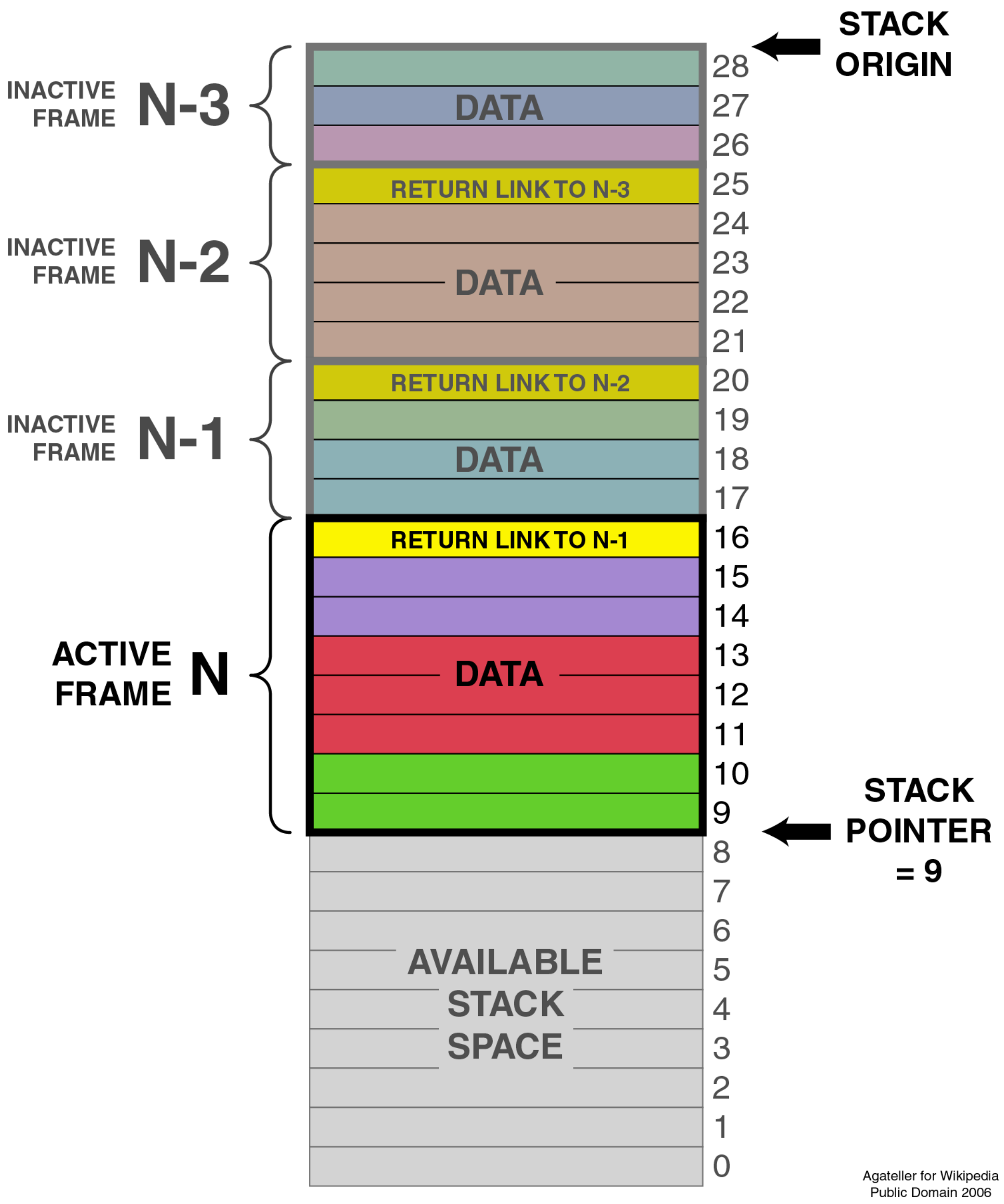
- Access to hardware
- No immediate feedback
- Control over testing
- More ability to fix bugs

Cons:

- Less Security
- No immediate feedback
- More bugs

Assignment 1

- Posted
- Choose your own partners
- I will send an email today for people who want to be assigned a partner
- Repos will be out after class; automatic testing will start tonight or tomorrow



Agateller for Wikipedia
Public Domain 2006

Where are things stored?

- First place: in CPU register, never in memory
  - Temporary variables like loop indices
  - Compiler decides this
- Second place: call stack
  - Small amount of dedicated memory to keep track of current function and local variables
  - Pop back to last function when done
  - Temporary!

Third place: the heap

- Very large amount of memory (basically all of RAM)
- Using new in Java or C++ puts variable on the heap
- We use malloc
    - Does not zero out memory. calloc does
    - C will not make you instantiate your variables
- Needs stdlib.h
- Returns pointer; don't need to cast to pointer type

Ways to store things

- Speed: registers > stack > heap
- Size: heap > stack > registers
- Longevity: heap > stack > registers

- Java rules work out well: store ”objects” and arrays on heap, just declare small “primitive types” and let the compiler work it out (Remember scope!!)

Allocation, pointers, and arrays

- What is an array?
- Can we use arrays without using array-like things?
  - Using pointers and malloc instead?
- Does this allow us to allocate arrays dynamically?

- Pointers and arrays are (mostly) *equivalent* in C

Memory leaks

- C does not have a garbage collector
  - Fast, efficient, you actually really want to be able to control this
  - But, obviously, huge pain and difficult to debug
- free() releases memory
  - Can be used for another variable
  - Not zeroed out
- Every malloc() should have a free()!
- After your program ends all memory is released

Memory leaks

- C does not have a garbage collector
  - Fast, efficient, you actually really want to be able to control this
  - But, obviously, huge pain and difficult to debug
- free() releases memory
  - Can be used for another variable
  - Not zeroed out
- Every malloc() should have a free()!
- **Every malloc() should have a free()!**
- After your program ends all memory is released

Segmentation faults

- Access “illegal” memory
  - Address that the OS didn’t give your program
- Given very very little information
- Debug using gdb (checkpoints, etc.)
- valgrind is useful for checking memory

Header files

- Generally end in .h
- Contain useful information
    - Function declarations
    - Structs, constants

Compiling and building

- Compile: convert code into machine-executable code
  - gcc –c [file name]
- Link: stitch together function calls between files
- Build: whole process
  - What gcc actually does when given file
  - Need to list compiled object files
- Student example

What happens when we change one file?

- Need to recompile that file
- Need to build final output file

- Can we do this automatically?

Makefile

- Lists dependencies
- Lists what you actually want to build
- Entire command: make
- If a file changes, compiles only what’s necessary
- make clean, make debug
- Very very useful!

In this class

- I will give you makefile
- Don’t need to change unless you use multiple files
  - You can, but probably won’t ever need to
  - Projects in this class are fairly small and self-contained

Variable types

- Int, long, etc. not necessarily the same on different systems
  - (If you use Windows long is likely 32 bits, but on Mac and Unix it's generally 64 bits)
  - (long long is 64 bits)
- Include stdint.h
- Unsigned (?)

Variable types

- Ints are OK for things like small loops
- If you care at all about size, should use int64_t
  - Fixed platforms means you don't NEED to for this class
  - (Except to handle function calls from test)
  - Good to get in the habit – guaranteed minimum size
- Unsigned is up to you
  - Controversial if they're a good idea

Variable types

- int64_t, int32_t
- uint64_t, uint8_t
- uint_fast64_t
- uint_least8_t

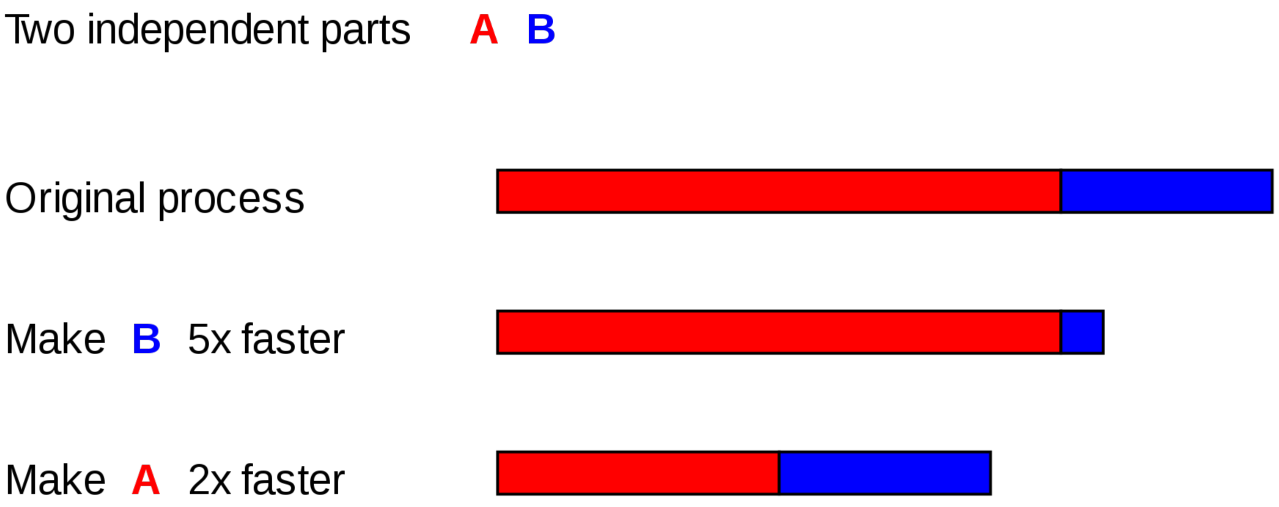
- INT64_MAX

Function pointers and sorting

Some simple efficiency principles

Time vs Space themed

Amdahl’s law



Two independent parts A B

Original process

Make B 5x faster

Make A 2x faster

- If a function takes up a p fraction of the entire program’s runtime, and you speed it up by a factor s, then the overall program speeds up by a factor

$$\frac{1}{1-p+\frac{p}{s}}$$

Amdahl's law and asymptotics

- If a portion of your program is asymptotically dominated by another, it is less likely to be worth speeding up

Cost of operations

- Adding? Multiplying? Floats? Ints?
- Dividing? Modulo?

Touching memory

Notes on how this works

- Allocation itself is essentially O(1)
- Writing to lots of places in memory is expensive
- How expensive is it?
  - Let's say we do a modulo, and an if, and a memory store (but in only one place)
  - Which is more expensive?
- Why am I cheating on the single "memory store"?

Function inlining

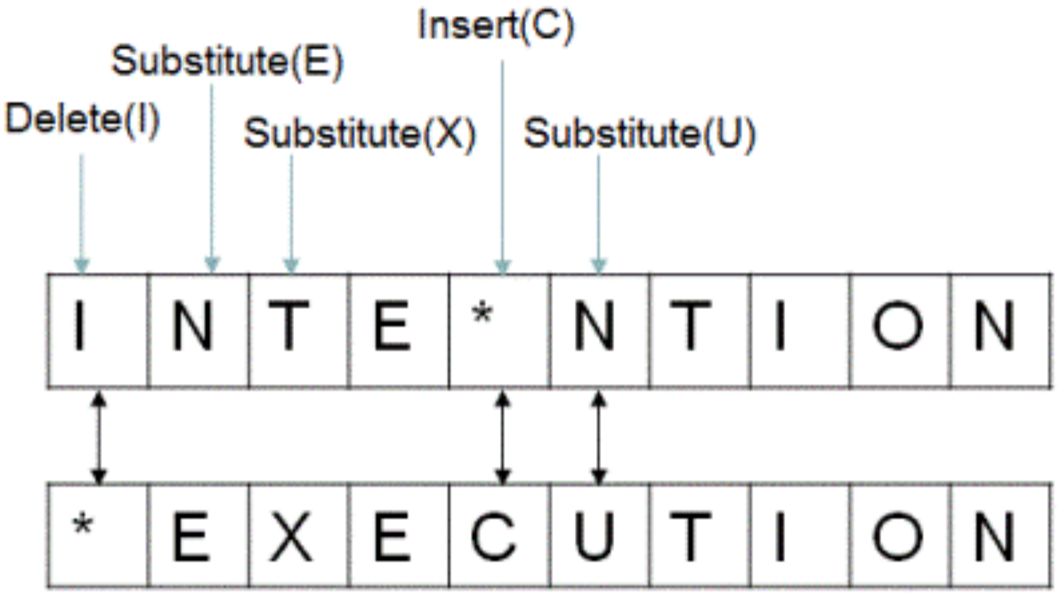
- Calling a function takes time (why?)
- With simple functions we can avoid that time
- To “inline” a function means to replace its contents in the code rather than doing a function call
- gcc will do this for you (and it’s really good at it)
- inline keyword: suggest to gcc that it should inline the function
    - Side effects in terms of linkage

Edit Distance

Problem

- Given two strings A, B
- Edit: insert, delete, replace (each costs 1)
- What is the minimum number of edits to get from A to B?

Example



Insert(C)

Substitute(E)

Delete(I)

Substitute(X)

Substitute(U)

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N

Algorithm: Dynamic Programming

- How can you build up edit distance recursively?

Analysis

- How much time does it take to calculate the edit distance between two strings of length n?
- How much space?

For next class

- How can we do this in $O(n)$ space?