CS 241 Honors
Git tutorial and surfing the high Cs

Kevin Hong, Aneesh Durg

University of Illinois Urbana-Champaign

January 29, 2018
Sailing the C-s

- Inspired by
  https://stackoverflow.com/questions/132241/hidden-features-of-c
It is *unlikely* that you’ll ever need this

- What is a branch prediction?
It is unlikely that you’ll ever need this

- What is a branch prediction?
- Where can I use this (compiler)?
It is unlikely that you’ll ever need this

- What is a branch prediction?
- Where can I use this (compiler)?
- What does it do?
It is unlikely that you’ll ever need this

- What is a branch prediction?
- Where can I use this (compiler)?
- What does it do?
- Has anyone ever used it?
  - Linux Kernel
It is *unlikely* that you’ll ever need this

- What is a branch prediction?
- Where can I use this (compiler)?
- What does it do?
- Has anyone ever used it?
  - Linux Kernel
- I’m better than a stupid machine right? Right?!?!
  - Manual branch prediction benefits from you knowing more about the program than the CPU
Okay, so the compiler is a genius

- GCC is a freaking beast
Okay, so the compiler is a genius

- GCC is a freaking beast
  - Loop unrolling
  - Tail call optimization
  - Instruction reordering

We cheated a bit in this example... Let's try something more fair next time around.
Okay, so the compiler is a genius

- GCC is a freaking beast
  - Loop unrolling
  - Tail call optimization
  - Instruction reordering
  - We cheated a bit in this example...
  - Let’s try something more fair next time around
Okay, so the compiler is a genius

- GCC is a freaking beast
  - Loop unrolling
  - Tail call optimization
  - Instruction reordering
  - We cheated a bit in this example...
  - Let’s try something more fair next time around

- Let’s take a look at some simple code
- Trying to outsmart the compiler can be bad
Abstract Art

- Trying to outsmart the compiler can be bad
  - We don’t need clever low level code as much
  - Duff’s device is an example
• Trying to outsmart the compiler can be bad
  • We don’t need clever low level code as much
  • Duff’s device is an example

• Abstractions are good for us
Abstract Art

- Trying to outsmart the compiler can be bad
  - We don’t need clever low level code as much
  - Duff’s device is an example

- Abstractions are good for us
  - Faster development cycles - more tools
  - Easier to reason about program
  - Easier to prove correctness
Abstract Art

- Trying to outsmart the compiler can be bad
  - We don’t need clever low level code as much
  - Duff’s device is an example

- Abstractions are good for us
  - Faster development cycles - more tools
  - Easier to reason about program
  - Easier to prove correctness

- We should think more about a program as a transformation of data rather than pushing bits around
Abstract Art

- Trying to outsmart the compiler can be bad
  - We don’t need clever low level code as much
  - Duff’s device is an example

- Abstractions are good for us
  - Faster development cycles - more tools
  - Easier to reason about program
  - Easier to prove correctness

- We should think more about a program as a transformation of data rather than pushing bits around

- C is a high level language?
Abstract Art

- Trying to outsmart the compiler can be bad
  - We don’t need clever low level code as much
  - Duff’s device is an example

- Abstractions are good for us
  - Faster development cycles - more tools
  - Easier to reason about program
  - Easier to prove correctness

- We should think more about a program as a transformation of data rather than pushing bits around

- C is a high level language?
  - It’s not assembly
Abstract Art

- Trying to outsmart the compiler can be bad
  - We don’t need clever low level code as much
  - Duff’s device is an example

- Abstractions are good for us
  - Faster development cycles - more tools
  - Easier to reason about program
  - Easier to prove correctness

- We should think more about a program as a transformation of data rather than pushing bits around

- C is a high level language?
  - It’s not assembly
  - You can implement OOP in C with function pointers and structs
Trying to outsmart the compiler can be bad
- We don’t need clever low level code as much
- Duff’s device is an example

Abstractions are good for us
- Faster development cycles - more tools
- Easier to reason about program
- Easier to prove correctness

We should think more about a program as a transformation of data rather than pushing bits around

C is a high level language?
- It’s not assembly
- You can implement OOP in C with function pointers and structs
- If we understand the compiler’s behavior, we can write code that seems less efficient, but is better (abstractions like vectorization that don’t change code)
Comma Operator

- So cool it’ll put you in a comma. (Get it, like coma?)
Comma Operator

- So cool it’ll put you in a comma. (Get it, like coma?)
  - Makes debugging easier
Comma Operator

- So cool it’ll put you in a comma. (Get it, like coma?)
  - Makes debugging easier
  - Can introduce interesting side effects in code
Comma Operator

- So cool it'll put you in a comma. (Get it, like coma?)
  - Makes debugging easier
  - Can introduce interesting side effects in code
  - Isn’t necessarily the most readable/intuitive
- Chains statements together
Comma Operator

- So cool it'll put you in a comma. (Get it, like coma?)
  - Makes debugging easier
  - Can introduce interesting side effects in code
  - Isn’t necessarily the most readable/intuitive
- Chains statements together
- Returns last value
λ functions are anonymous functions that represent computation in λ-calculus.

λ-calculus? I thought calc 3 was bad enough...

Competing representation with Turing Machines
Proved equivalent
Has nothing to do with the real world - this is good, because we’re bad at reality

Underlying basis for functional programming
No side effects
"Easy" to understand code
Functions as first class objects (almost like functions pointers)
λ functions are anonymous functions that represent computation in λ-calculus.

λ-calculus? I thought calc 3 was bad enough...
Lambdas in C (or why I have a broken heart)

- λ functions are anonymous functions that represent computation in λ-calculus.
- λ-calculus? I thought calc 3 was bad enough...
  - Competing representation with Turing Machines
λ functions are anonymous functions that represent computation in λ-calculus.

λ-calculus? I thought calc 3 was bad enough...
- Competing representation with Turing Machines
- Proved equivalent
λ functions are anonymous functions that represent computation in λ-calculus.

λ-calculus? I thought calc 3 was bad enough...
- Competing representation with Turing Machines
- Proved equivalent
- Has nothing to do with the real world - this is good, because we’re bad at reality
λ functions are anonymous functions that represent computation in λ-calculus.

λ-calculus? I thought calc 3 was bad enough...

- Competing representation with Turing Machines
- Proved equivalent
- Has nothing to do with the real world - this is good, because we’re bad at reality

Underlying basis for functional programming
λ functions are anonymous functions that represent computation in λ-calculus.

λ-calculus? I thought calc 3 was bad enough...
- Competing representation with Turing Machines
- Proved equivalent
- Has nothing to do with the real world - this is good, because we’re bad at reality

Underlying basis for functional programming
- No side effects
Lambdas in C (or why I have a broken heart)

- $\lambda$ functions are anonymous functions that represent computation in $\lambda$-calculus.
- $\lambda$-calculus? I thought calc 3 was bad enough...
  - Competing representation with Turing Machines
  - Proved equivalent
  - Has nothing to do with the real world - this is good, because we’re bad at reality
- Underlying basis for functional programming
  - No side effects
  - ”Easy” to understand code
Lambdas in C (or why I have a broken heart)

- λ functions are anonymous functions that represent computation in λ-calculus.
- λ-calculus? I thought calc 3 was bad enough...
  - Competing representation with Turing Machines
  - Proved equivalent
  - Has nothing to do with the real world - this is good, because we’re bad at reality
- Underlying basis for functional programming
  - No side effects
  - ”Easy” to understand code
  - Functions as first class objects (almost like functions pointers)
Lambdas in C (or why I have a broken heart)

- Only supported in GCC (not portable!)

Kevin Hong, Aneesh Durg (UIUC)
Gitting Your Feet Wet
January 29, 2018
Lambdas in C (or why I have a broken heart)

- Only supported in GCC (not portable!)
- I'm forever seeking closure...
Lambdas in C (or why I have a broken heart)

- Only supported in GCC (not portable!)
- I’m forever seeking closure...
  - Not really a lambda function - no closures, not anonymous
Lambdas in C (or why I have a broken heart)

- Only supported in GCC (not portable!)
- I'm forever seeking closure...
  - Not really a lambda function - no closures, not anonymous
  - Memory model get in the way...
Lambdas in C (or why I have a broken heart)

- Only supported in GCC (not portable!)
- I’m forever seeking closure...
  - Not really a lambda function - no closures, not anonymous
  - Memory model get in the way...
  - Intended to fulfill a different purpose
Lambdas in C (or why I have a broken heart)

- Only supported in GCC (not portable!)
- I’m forever seeking closure...
  - Not really a lambda function - no closures, not anonymous
  - Memory model get in the way...
  - Intended to fulfill a different purpose
- Lexical Scoping!!!
Only supported in GCC (not portable!)
I'm forever seeking closure...
  Not really a lambda function - no closures, not anonymous
  Memory model get in the way...
  Intended to fulfill a different purpose

Lexical Scoping!!!
  Not exactly a closure, but the next best thing
LPT: be *assertive*

- Compile time asserts?
LPT: be *assertive*

- Compile time asserts?
  - Faster development cycles
Compile time asserts?
- Faster development cycles
- Catch errors before running the program!
LPT: be assertive

- Compile time asserts?
  - Faster development cycles
  - Catch errors before running the program!
- Almost reminiscent of static type checking
Compile time asserts?
  - Faster development cycles
  - Catch errors before running the program!

Almost reminiscent of static type checking

Useful to determine if code will run on computers with limited resources
This slides have no typeofs

• GCC only - again
• GCC only - again
• Familiar if you’re coming from python
This slides have no typeofs

- GCC only - again
- Familiar if you’re coming from python
  - But not as useful
- GCC only - again
- Familiar if you’re coming from python
  - But not as useful
- Can be used to implement templetized code
• GCC only - again
• Familiar if you’re coming from python
  • But not as useful
• Can be used to implement templateized code
• Can combine this with some macros (_Generic) we’re not going to cover tonight that allow for much greater control
Humans are bad at optimizing low level code
Vitamin C is good for you.

- Humans are bad at optimizing low level code
- GCC C has some pretty neat high level capabilities
Humans are bad at optimizing low level code
GCC C has some pretty neat high level capabilities
  makes sense when you think about how much the world needs C
Humans are bad at optimizing low level code
GCC C has some pretty neat high level capabilities
- makes sense when you think about how much the world needs C
Getting easier to right code that lets you think at a high level but control all the little details
...but too much of a good thing is a bad thing

Choose the right language for the right task
Choose the right language for the right task

- If your task is systems programming, you’re (probably) in the right place
Choose the right language for the right task

- If your task is systems programming, you’re (probably) in the right place

- Not the most portable
Choose the right language for the right task
  - If your task is systems programming, you’re (probably) in the right place
- Not the most portable
- Not the most readable
Choose the right language for the right task
  - If your task is systems programming, you’re (probably) in the right place
- Not the most portable
- Not the most readable
- Too many high level features, and you lose low-level control...