C Wrapup

CSE 333 Winter 2021

Instructor: John Zahorjan

Teaching Assistants:

Matthew Arnold	Nonthakit Chaiwong	Jacob Cohen
Elizabeth Haker	Henry Hung	Chase Lee
Leo Liao	Tim Mandzyuk	Benjamin Shmidt
Guramrit Singh		

Programming Languages

- How quickly can I write a correct program?
  - Expressiveness
  - Available libraries?
- How hard is it to write an (obviously) incorrect program?
  - "Language analysis"
- How efficient is the executable?
  - Algorithmic efficiency
  - Code efficiency
- How portable is my program?
  - Different hardware? Different OS?

More Programming System Considerations

- How hard is it to interact with code written in other languages?
- How well does language support teams of programmers?
  - How well does development environment support teams?
  - How well does build support teams?
- Language and parallel/distributed execution?
- Portability of executable?
  - Virtual machines
  - Containers

Programming Languages

	Assembler	C	Java
Expressiveness	0	4	8
Language analysis	0	4	7
Executable Efficiency	8	9	6
Portability	0	6	10
Flexibility	9	8	4

Sum 1000000 random long's 1000 times:

- Assembler – sorry, didn't implement it
- C – 0.293 seconds
- Java long – 0.466 seconds
- Java Long – 13.553 seconds

Programming Languages (Aside)

	Assembler	C	Java	C++
Expressiveness	0	4	7	8.5
Language analysis	0	4	7	6.5
Executable Efficiency	8	9	6	10
Portability	0	6	10	8.5
Flexibility	9	8	4	7

Sum 1000000 random long's 1000 times:

- Assembler – sorry, didn't implement it
- C – 0.293 seconds
- **C++ - 0.289 seconds**
- Java long – 0.466 seconds
- Java Long – 13.553 seconds

C Expressiveness

- You get memory allocation (static/heap/global variables)
- You get basic control flow (loops)
- You get functions / procedures
- You get primitive support to generate specialized code (preprocessor)

- Single, global name space (for global vars and all functions)
- No language support for program structuring beyond procedures
- No language support for generics
- No (real) language support for information hiding
- No memory management

Language Analysis

- C is intended to allow (near direct) access to hardware
  - Can operate "below the semantics of the language" to directly modify memory, for instance
- That makes program analysis difficult
  - What could a pointer be pointing at in this line of code?

- The compiler thinks pretty much every syntactically correct statement is semantically correct
- The language tolerates (embraces?) that many things that can be said, legally, have undefined result

Execution Efficiency

- A general lesson: simplest is fastest
  - C makes close to no promises (vs. Java…)
- There are no run-time checks, unless you program them
  - There is no run-time interpreter
  - "All the action" is static (at compile time)
- The optimizer is very good at what it does
  - Constant propagation
  - Re-ordering code
  - Dead code elimination

Portability

- C standards
- Standard library / system calls

- App code must be recompiled on target system
- App code must be linked with implementation of standard functions written for target system

Portability

- Things That Go Wrong
  - Library functions, including std lib functions, don't exist or have different semantics
    - especially with reflecting errors
  - The program has hardware dependences
    - E.g., size of a long int
    - size of a pointer
    - addresses that indicate stack allocation (vs. heap)
  - Code has a bug that is benign on original system but un-benign on new system
    - E.g., write past end of an array
    - Note: this can make code non-portable from one version of compiler/language to the next!
- Example: Why does course project require gcc 9?

Flexibility

- Can I integrate my code written in some language with code written in another?
  - Sure – files, text output, etc.

- C is a de facto lowest common denominator
  - Languages often have tools that let them interact with (and so use) libraries written in C