Course Intro

ITEC320

author nlahn@radford.edu; modified by ibarland@radford.edu.

What’s the point of this course?

- “Procedural Analysis and Design”?
  - What does it mean?
  - Why is it important?

author nlahn@radford.edu; modified by ibarland@radford.edu.

Procedural Analysis and Design

- Procedural programming is a type of programming *paradigm* (i.e., style)
  - Loops, if statements, procedures / functions
- Most of your prior classes focused on the Object Oriented paradigm, using Java
- We will be using a programming language called “Rust”, which is “more procedural” and “less OOP”
- We will make many comparisons between Rust and Java, looking at how they are similar and different

author nlahn@radford.edu; modified by ibarland@radford.edu.

Procedural ***Analysis*** and Design

- For *any* part of a program, you *must* be able to:

1. Break it down into smaller parts (if possible), and recursively analyze each part
2. Describe *why* it is important
3. Know how to *apply* it in different circumstances
4. *Evaluate* it in terms of *correctness*, *efficiency*, and *maintainability*, and *style*

Never settle for "I'm not sure".

Procedural Analysis and ***Design***

- Given a computing problem, you should be able to:

1. Carefully read the problem statement; clarify ambiguities
2. Brainstorm a set of approaches; identify concepts you will need
3. Write down a summary of your approach, using **pseudocode**
4. Evaluate whether your approach works, and fix flaws before implementing it! Redesign as needed
5. Select appropriate programming language features for the task
6. Implement your design
7. Test your design

Course Topics

- Static versus dynamic error checking / decision making
- Type systems
- Module design and encapsulation
- Generic programming
- Polymorphism
- Pointers and references
- Memory management
- Programming language design tradeoffs
- Testing programs
- How to think

author nlahn@radford.edu; modified by ibarland@radford.edu.

Our main goals

- Learn about how programming languages are designed and structured
- Learn about different paradigms (styles) of coding
- Learn about what's happening under the hood when your program runs.
- Learn about strategies for code maintainability and reliability
- Learn how to problem solve, writing logically complex blocks of code

author nlahn@radford.edu; modified by ibarland@radford.edu.

Java

- About 30 years old
- Iconic OOP language
- Memory is managed automagically using a garbage collector
- Slower than systems programming languages
- High level
- Little control over low level functionality

Rust

- About 10 years old
- Multi-paradigm language
- Memory management uses the “ownership” model
- Low-level control
- Very fast
- Very strict compiler
- Focus on reliable programming (secure; few vulnerabilities)
- Great for multi-threading

author nlahn@radford.edu; modified by ibarland@radford.edu.

A brief history of programming languages

Just a few of them….

author nlahn@radford.edu; modified by ibarland@radford.edu.

In the beginning…

01100100110101010101010010101
01001010101010010101010101001
01010100101010101001010101010
10010101010010101010100101010
10101001010101001010101010010
10101010100101010100101010101
00101010101010010101010010101
01010010101010101001010101001
01010101001010101010100101010

There was machine code

Each instruction was very simple.

Raw binary. Not easily readable.

All the power to do basically anything you want.

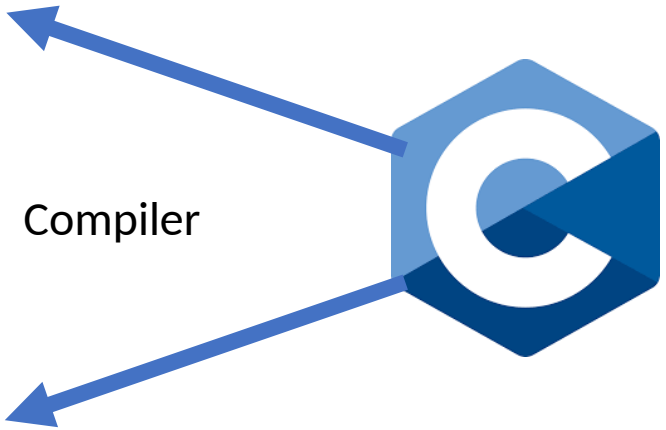
Enter assembly

- Each instruction maps to one binary machine code operation
- A program called an “assembler” can convert assembly into machine code
- Human readable (sort of)

```
                        ; Output all even numbers from 1 to 100

    mov ecx, 1          ; initialize our counter
loop_start:
    mov eax, ecx        ; Copy the current value in ecx to eax
    and eax, 1          ; Is least significant bit is set? (odd #)
    jnz not_even        ; Jump to not_even if the result non-zero

    ; Print even number
    mov eax, 4          ; syscall number for sys_write
    mov ebx, 1          ; file descriptor 1 (stdout)
    mov edx, 3          ; length of the message
    mov ecx, ecx_msg    ; address of the message to print
    int 0x80            ; interrupt to invoke the syscall

not_even:
    inc ecx             ; Increment our counter
    cmp ecx, 101        ; Compare ecx with 101 (end of range)
    jle loop_start      ; Jump back to loop_start if ecx <= 100
```

The C programming language (1972)

Assembly / Machine code For Windows

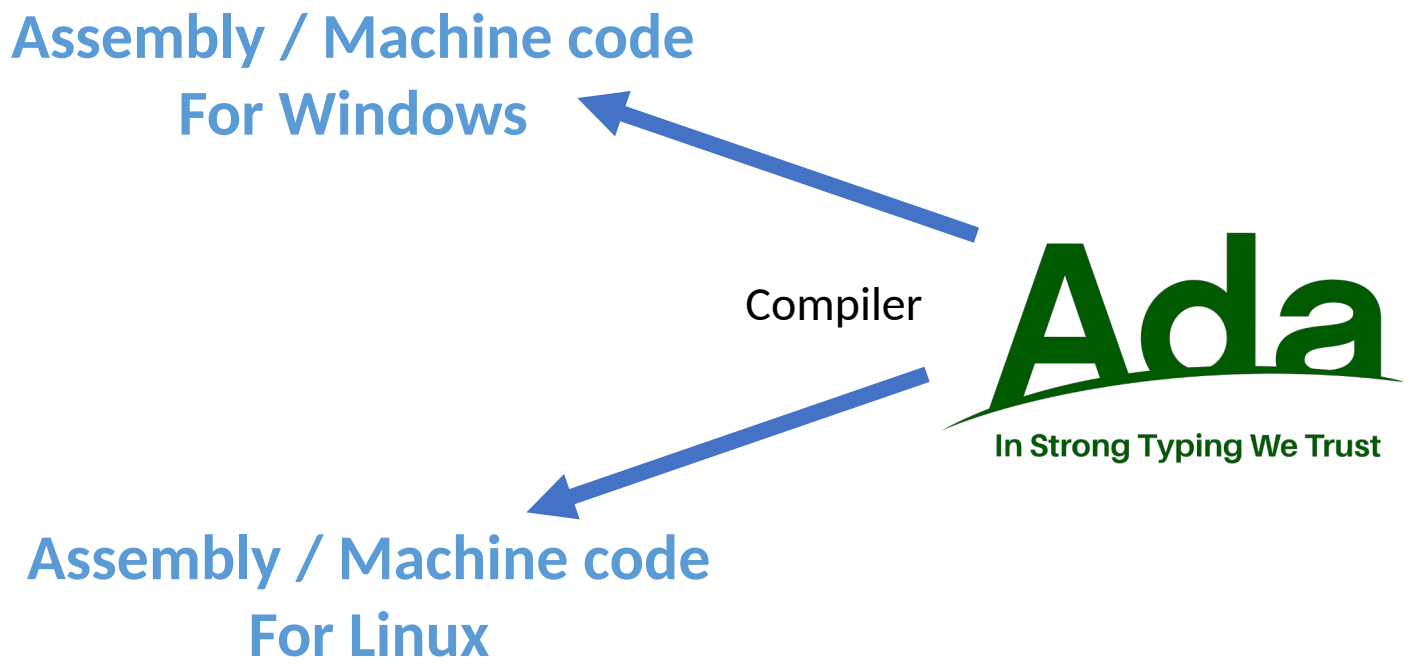


Assembly / Machine code For Linux

```
#include <stdio.h>

int main() {
    int i = 1;
    while (i <= 100) {
        if (i%2 0= 0) printf("%d\n", i);
    }
    return 0;
}
```

author nlahn@radford.edu; modified by ibarland@radford.edu.

The Ada programming language (1980)




```
with Ada.Text_IO;

procedure Even_Numbers is
begin
    for i in 1..100 loop
        if i mod 2 = 0 then
            Ada.Text_IO.Put(Item => i);
            Ada.Text_IO.New_Line;
        end if;
    end loop;
end Even_Numbers;
```

author nlahn@radford.edu; modified by ibarland@radford.edu.

Ada vs. C

One of these days you're going to shoot yourself in the foot!

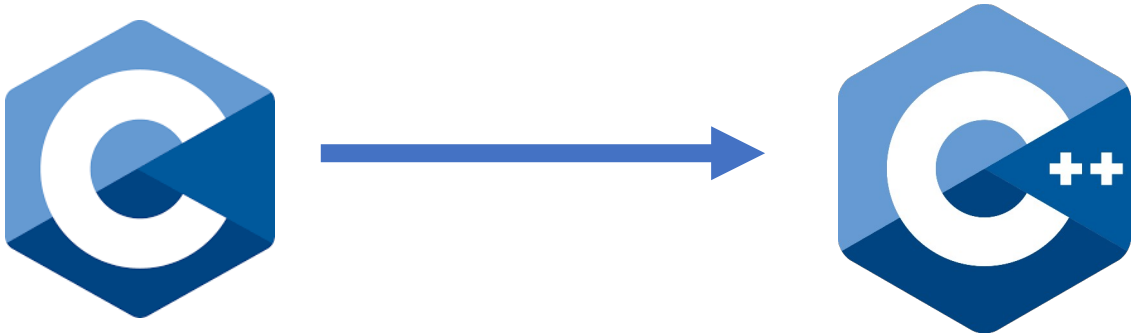
At least I don't have to jump through a ton of hoops to do *anything*

Reliability

Flexibility

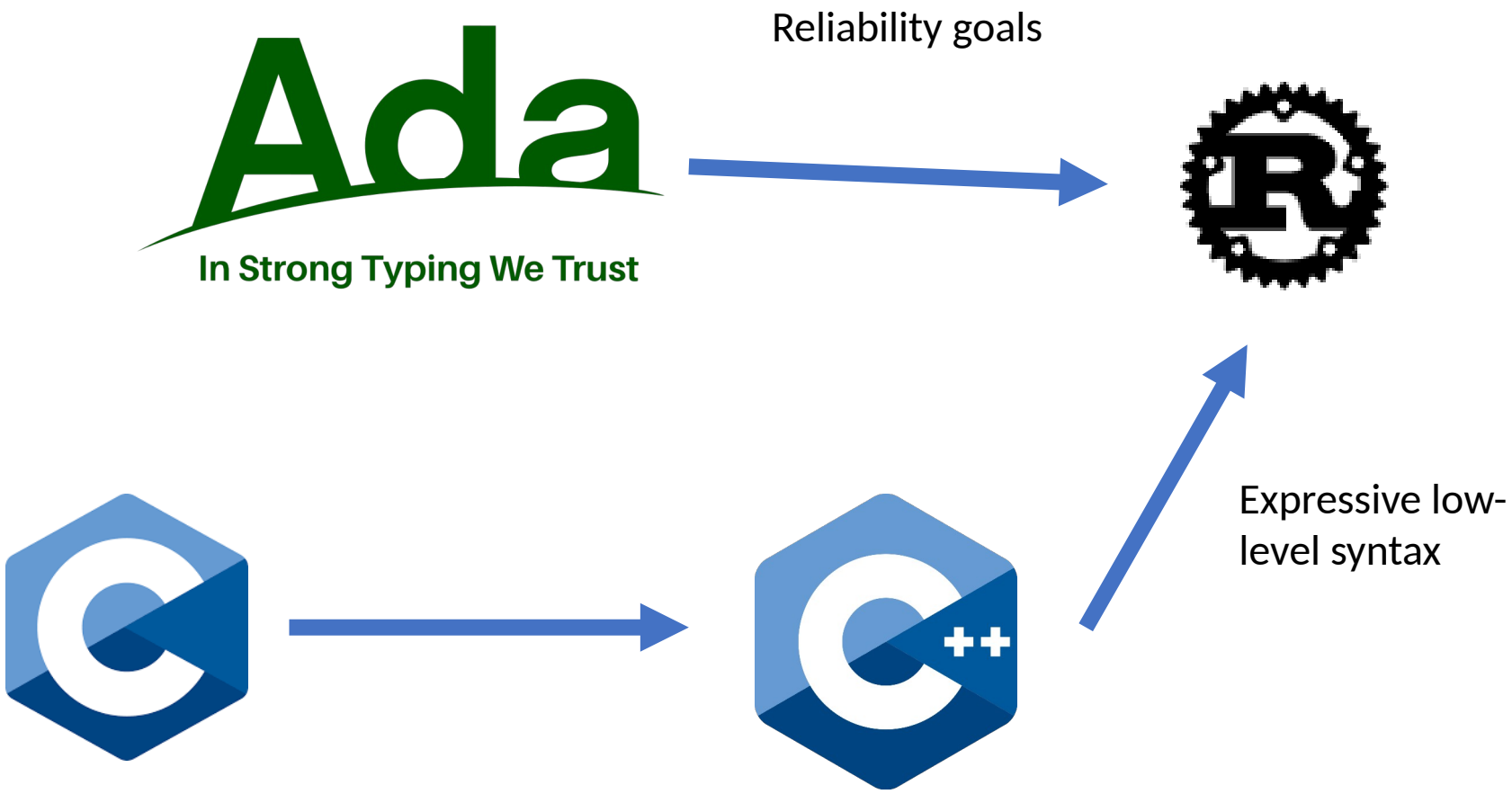
author nlahn@radford.edu; modified by ibarland@radford.edu.

C++ : C with OOP features (1983)



author nlahn@radford.edu; modified by ibarland@radford.edu.

Rust : Can we make reliable and flexible programs?



If you don’t know how things work, then your program will probably not compile.

So you really have to understand how stuff works to get anywhere.

author nlahn@radford.edu; modified by ibarland@radford.edu.

Rust in Industry

- The main use case of Rust is low-level systems programming
- Google, Meta, AWS, Huawei, and Microsoft are all strong supporters of the Rust Foundation
- Google is using Rust for Android code
- AWS wrote “Firecracker” in Rust
  - Firecracker is a light-weight VM that powers AWS Lambda, a pinnacle of AWS cloud architecture
- Rust has recently been approved for inclusion into the Linux kernel

Rust Resources

- Rust by example : https://doc.rust-lang.org/rust-by-example/
- The Rust book : https://doc.rust-lang.org/book/
- The Rust reference : https://doc.rust-lang.org/reference/
- The Rustonomicon : https://doc.rust-lang.org/nomicon/

We will be drawing heavily upon materials from the Rust book. I recommend reading all of it.

author nlahn@radford.edu; modified by ibarland@radford.edu.

Syllabus Discussion

- See the syllabus document posted on D2L