General-Purpose Languages: What Are Your *Habits*??

Defcon 31

BIC Village - Blacks in Cyber

D.J. Davis, A+, CISSP

(ZeroRingDefender)

August 12, 2023

Version 1.0

Objectives

How this presentation came about...

At DC30 I Listened to a *lot* of vulnerabilities in code. I sank in my chair with a defeated feeling of how all of us are so MESSED UP by the vulnerabilities... ... I wondered: Which Language SHOULD We Use ?????

Objectives:

- Explore some vulnerabilities that can occur in **languages** (not OSes; not configuration)
  - Outline a few ways to create less buggy/vulnerable code
- Review some popular G-P Ls, some real-world characteristics, and security vulnerability considerations
  - Examine ideas to help us choose our languages (*Always* have a *current* language!)
  - Provide a "Hello, World!" example, and information to start programming in these languages (to gen up a Linux laptop, *see the Appendix online*)

Prerequisites

1) Have looked at a program source code listing before
2) Basic knowledge of program variables
3) Awareness of Compiled programs vs Interpreted programs

With this said, in this presentation, I believe there is something meaningful for everyone

Disclaimer --

This presentation is not affiliated with my work. These are my own personal ideas. Feel free to choose and use selectively.

Our Game Plan … … …

- Outline some General-Purpose Languages
- Learn from some behaviors of legacy languages
- Discuss 12 current G-P Ls and outline some of their behaviors
- Outline some considerations in choosing our language
- View the types of software vulnerabilities that can occur in languages
- Consider some Recommendations from my Observations in Vulnerability Database

~~ Kindly defer questions to the end of the presentation

About D.J.

Background:

Ops, Dev, Mainframe Eng, Network Eng, WAN Design Eng, IT Integration, Sustaining Eng, Computer Security

- Locale:

  Live/work/play in Washington DC region

- My humble beginnings… …

  COBOL/CICS, EasyTrieve Plus, Business BASIC (Electronic Cash Register PC, police/dispatch Midrange)

- Alma Mater:

  Virginia Commonwealth University - Go Rams!!

  BS Business, Info Sys; MS Business, Info Sys - IT Management

some Old-school LEGACY G-P Ls

Fortran - 1957 - **First compiler**. Math / science / engineering. **First 3GL**

Lisp - 1958 - **First interpreter**. Math, AI research. **Second-oldest 3GL**

ALGOL - 1958 - compiler. General computing. **Father of several languages**

COBOL - 1959 - compiler. **Business record** processing. **IBM mainframes**

BASIC - 1964 - interpreter. Teaching, PCs, Mid-range, home/commercial use

PL/1 - 1964 - compiler. Business (COBOL) and scientific (Fortran) use. **IBM mainframes**

APL - 1966 - interpreter. Math, finance, AI, image manipulation

Pascal - 1970 - compiler. Teaching, commercial use

some CURRENT G-P Ls

C	- 1972	- compiled
Objective-C	- 1984	- compiled
C++	- 1985	- compiled
Perl	- 1987	- interpreted
Python	- 1991	- interpreted
JavaScript	- 1995	- interpreted
PHP	- 1995	- interpreted
Ruby	- 1995	- interpreted
Java	- 1995	- compiled to Java bytecode
C#	- 2000	- compiled to .Net IL bytecode
Kotlin	- 2011	- compiled to Java bytecode
Go	- 2009	- compiled
Rust	- 2010	- compiled
Swift	- 2014	- compiled

Initial growth of the World Wide Web

How We Started - LEGACY G-P Ls

Let's Go Old-school For 4 Minutes... ...

Note: Older languages are still available today and have evolved

Fortran is a popular language on supercomputers

Several legacy languages have OO versions

A language can be set up as both a compiler and an interpreter (e.g. APL, BASIC)

BASIC - 1964 - interpreted

In 1970s/80s versions, the interpreter also serves as a text editor

Upon entering a program line, the interpreter tokenizes keywords to save on memory space and increase program execution speed. Originally a teaching language; widely-used on PCs and mid-range systems

Program terminates on invalid I/O / div by zero / subscript overrun / invalid variable unless the language variant implemented error trapping

Historically, variable types are floating point numbers and strings

Input statement variable is a string; variable length limited to 255 chars (max string length)

Does not overrun variables

FUN FACT: BASIC was planned as a compiled language but memory footprint was too large; switched to interpreted language

```
10 REM a comment - HELLO.BAS
15 LET A = 5 : 'Multi-statement line; LET keyword is optional
16 B = 6
17 A$ = "" : 'String variable
20 PRINT "Hello, World!"
25 INPUT$ "Press ENTER to End Program ", A$
30 END
```

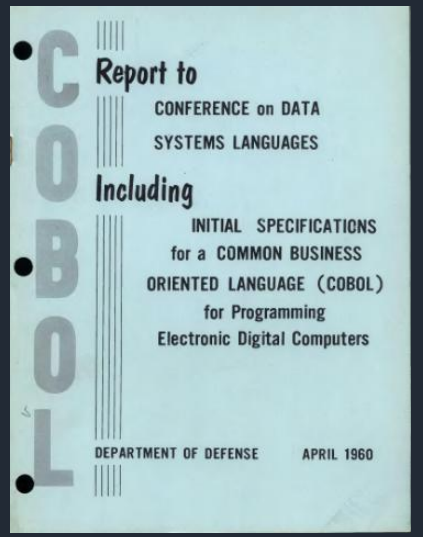
COBOL – 1959 - compiled

Primarily for business processing, file I/O. Very limited math support: + - * / ** ( )
Most often seen on IBM mainframes; most often used unit record I/O (fixed-length records)
Program abends (terminates) on invalid I/O / div by zero / invalid mem reference
Generally does not overrun variables (fixed-length variables; compiler/runtime checks)

```
000001* A COMMENT - HELLO.COB
000002  IDENTIFICATION DIVISION.
000003      PROGRAM-ID. HELLO.
...
       ENVIRONMENT DIVISION.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
         01 WS-TEST-STRING.
            03 WS-PART1   PIC X(16)  VALUE "THIS IS A TEST. ".
            03 WS-PART2   PIC S9(5)V99  VALUE 8.99 COMPUTATIONAL-3.
         01 WS-INPUT-LINE PIC X(15) VALUE SPACES.

      PROCEDURE DIVISION.
         DISPLAY "HELLO, WORLD!".
         ACCEPT WS-INPUT-LINE.
         STOP RUN.
```



A major Contributor to COBOL
Rear Admiral Grace M. Hopper, USN

Pascal - 1970 - compiled

Based on ALGOL 60 (1960), Pascal was a breakaway project from an ALGOL rewrite

Originally regarded as a teaching language; moderate success until the popularity of C increased in academic and commercial settings

Structured language

Was used to write Apple Lisa OS and early Macintosh OS

In 1981 Brian Kernighan wrote a paper describing the notable deficiencies of Pascal as compared to C

```
// a comment - HELLO.PAS
program Hello ;
begin
  var inputString: string;
  writeln ('Hello, World!');
  readln (inputString);
end.
```

The *Apple Lisa OS is now available online for download at the Computer History Museum*

Photo courtesy Timothy Colegrove

PL/1 - 1964 - compiled

Designed primarily by IBM

At the time, a large language for both I/O and math processing. Performs tasks handled by both Fortran and COBOL. IBM used PL/1 as a systems language

Program abends (terminates) on invalid I/O / div by zero / invalid memory reference

In general usage, it does not overrun (fixed-length) variables

“Modern features” (e.g. dynamic mem allocation, Pointers including pointer arithmetic, multitasking)

Notice the line: Procedure Options Main. Pl/1 had an influence on C

D.J.'s first language; much more compact than COBOL; shout-out to Dr. James Ames (VCU CS ret.)

```
      /* A COMMENT - HELLO WORLD IN PL/1 */
HELLO:   PROCEDURE OPTIONS (MAIN);
         PUT SKIP DATA ('HELLO, WORLD!');
         STOP;
END HELLO;
```

Some Themes from Legacy G-P Ls

- Legacy programs might be less vulnerability-prone on legacy IBM mainframe due to unit record I/O, 80 / 96 character card input, fielded 3270 screens
- BASIC interpreter limits/enforces input length and string variable length
- Programs terminate on divide by zero, subscript out of bounds, I/O error, bad data, invalid mem reference
- Legacy languages are generally case-INsensitive
- Legacy languages evolved over time but not as quickly as today
- Problems were more in the Bug category and less in the Vulnerability category because systems were less connected and more protected during input (Every bug is ***not*** a vulnerability … …)
- Without network connectivity / Internet / API Methods, input was from keyboard / disk / tape and did not need to be sanitized as much
- Behaviors to note in legacy languages:
  - OS and Language controls were on effected program input and on Variables to prevent data overruns
  - Languages did not have memory Pointer variables, or they were used less often
  - Program execution stopped upon invalid references, out-of-bounds subscripts, data overruns, etc
  - Any APIs were accessible on the same system via programming; usually not remotely

Some language characteristics to observe …

Characteristics that are beneficial to observe …

- Minimum program size
- Compiled or interpreted language
- Compile speed, Execution speed
- Semi-colon requirement as statement separators
- Input handling, ease of input; risk of input (buffer overflow)
- Can we overrun variables?
- Behavior with unused variables -- Must every defined variable be used?
- Concurrency -- Single-threaded vs concurrent processing
- Error detection mechanisms
- Variable types
- Language is Case-sensitive / Case-insensitive - Perhaps ALL modern languages are Case-sensitive

```
void ourHumorForToday  ()  {
        bMorning    =  True ;
        bAfternoon =  False ;
        bNoon         =  Maybe ;
}
```

C - 1972 - compiled (#1 of 12)

C is a ubiquitous, omnipresent, mid-level, systems-oriented language

Produces the fastest code outside of Assembly

Compact, port-able, able to access HardWare (pointers, pointer manipulation, inline Assembly)

Gives power AND RESPONSIBILTY to programmer. NO BOUNDS CHECKING. Can overrun variables; C does not check length

Operating systems and most language compilers are written in C and/or C++

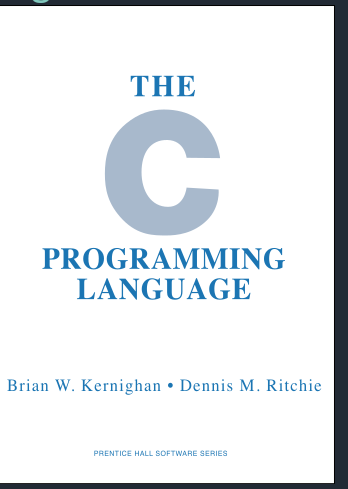
PL/1 influenced C

The C Library (libc / glibc) has similarities to Linux system calls

libc, when printing with buffered I/O, can run faster than handwritten Assembly that does not perform buffering

```
/* a comment - hello_c.c    */        // also a comment
// compile with: gcc hello_c.c -o hello_c

#include <stdio.h>
int main ()
{
   printf ("Hello, World! \n");
}
```



C's behaviors

(ref)

- Compiled pgm size for Hello World: 24 kB
- OO Model: None
- Semi-colon requirements as stmt terminator: Required

- Safer input into String objects; or routines that limit input: No
- Concurrency (Single-threaded vs concurrent processing): Can start new threads via function calls
- Program behavior on Divide by zero: Stop / Fatal error
- Garbage Collector (for objects): No, N/A
- Memory safe: No
- Thread safe: No
- DJ's Difficulty Index: **6 out of 10**
- Language website: Search: C programming language

- Frameworks: Kore, Vely, facil.io
- Variable types: Integers (diff sizes, signed/unsigned), Floats, Char arrays, no booleans
- Behavior with unused variables (must use every variable defined?): No
- Can overrun variables: Yes
- Error detection mechanism: None. Check the return value of a function call
- Program behavior on Invalid File I/O: Continue
- Pointer safe: No
- Variable safe (overwrite, wrap-around, buff overruns): No
- Goto statement: Yes
- Release cycle: Several times per year for most compilers. glibc (GNU C library) every 6 months. C17 standard on June 2018. C23 due in 2024; draft is online

C++ - 1985 - compiled (#2 of 12)

Originally named C With Classes. Extended C with OO and numerous other features

Very complex language, Extensive standard library, Extensible framework with Standard Template Library (STL)

Most games and game engines are written in C++

Many modern-day language features appear to come from C++: Objects / Classes / Methods, Generics, Templates, Constructors, Lambdas, Namespaces, References, Semantics

Multiple ways to do a task in C++; Additionally, there is the C-style of programming. C++ should compile any C program

C++ techniques expand / extend with C++14, C++17, C++20. C++ appears to undergo extensive changes in new versions

In early years, language incompatibilities existed in different C++ implementations. C++98 received ISO standardization but Standard Template Library feature was incompatible with GNU and Microsoft compilers. Eventually Microsoft and GNU had to rewrite their compilers. Changes were made to Linux kernel (written in C) because C++ couldn't compile it

Linus Torvalds is not keen on having C++ code added to Linux kernel

```
// a comment - hello_c.cpp        /* another comment - hello_c.cpp  */
// compile with: g++ hello_cpp.cpp -o hello_cpp
#include <iostream>
using namespace std;
int main ()
{
   cout << "Hello, World!" << endl;
}
```

C++'s behaviors

(ref)

- Compiled pgm size for Hello World: 25 kB
- OO Model: **Yes; requires developer to manage lifetime of objects**
- Semi-colon requirement as stmt terminator: Required
- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): Can start new threads via function calls
- Program behavior on Divide by zero: Stop / Fatal error
- Garbage Collector (for objects): **No**
- Memory safe: No
- Thread safe: No
- DJ's Difficulty Index: **10 out of 10**
- Language website: isocpp.org
- Frameworks: Boost, CLang, Qt, Wt
- Variable types: Integers (diff sizes, signed/unsigned), Floats, Char, String **object**, **Boolean**
- Behavior with unused variables (must use every variable defined?): No
- Can overrun variables: Yes
- Error detection mechanism: try/catch code block to detect error
- Program behavior on Invalid File I/O: Continue
- Pointer safe: No
- Variable safe (overwrite, wrap-around, buff overruns): No
- Goto statement: Yes
- Release cycle: Several times per year for most compilers and for libc++ (C++ library). C++20 standard on December 2020. C++23 due TBD; draft is online

Python - 1991 - interpreted (#3 of 12)

Easier language that has taken the role of BASIC as a “first” language -- Widely used to learn programming

Used for applications, middleware, integration, data analytics, ML/AI

Popularity/flexibility comes from lots of core and 3rd-party libraries that are invoked with "import“ statement

Python interpreter has built-in debugger (pdb)

Python versions 2 and 3. Use version 3. Quick way to tell… v2: print “Hi” v3: print (“Hi”)

For new/changed program, interpreter stores translated byte-code in hidden subdirectory (__pycache__)

Indentation alignment rules can cause issues (one space too few/many and program does not run)

Oddness in the way its coded; some from PEP

Before a function is called, the interpreter must have already parsed the function. Generally, a line of code cannot call a function below it (but there is a caveat). Because of all this, the starting point of the program is generally at the BOTTOM of a source file

If a person gets access to a computer (esp. Linux) Python is more likely to be installed. Can’t run netcat? Type in your own!

```
#!/usr/bin/python3
# (on Linux the previous line specifies the interpreter)
# a comment - hello_py.py
# To run on Linux:  ./hello_py.py
# To run on Windows and Linux:  python3 hello_py.py
print ("Hello, World!")
```

Python's behaviors (ref)

- Compiled pgm size for Hello World: n/a
- OO Model: **Yes, optional**
- Semi-colon requirement as stmt terminator: **No. Line break is stmt terminator**

- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): Std lib has Threading modules; Global Interpreter Lock allows concurrency but not multiprocessing
- Program behavior on Divide by zero: Stopped by interpreter
- Garbage Collector (for objects): **Yes**
- Memory safe: **Yes**
- Thread safe: **Yes**
- DJ's Difficulty Index: 3 out of 10
- Language website: python.org

- Frameworks: Django, Flask
- Variable types: Number, String, Boolean, Object
- Behavior with unused variables (must use every variable defined?): No
- Can overrun variables: **No**
- Error detection mechanism: try/except code block
- Program behavior on Invalid File I/O: **Continue**
- Pointer safe: **Yes / n/a**
- Variable safe (overwrite, wrap-around, buff overruns): Yes
- Goto statement: No
- Release cycle: Minor release every October; Point releases every 2 months for duration of 18 months; some security fixes after this

JavaScript - 1995 - interpreted (#4 of 12)

Typically runs on **client side** by web browser to provide dynamic effects to a web page

Is usually tied closely to web browser for web pages; not standalone programs on the computer

Program can be embedded in HTML or called from HTML

Can be run from command line (and as backend program) with Node.js

To run in Linux: node program.js

```
// a comment  - hello_js.js
/* another comment */
// JavaScript code is embedded in HTML file or called from HTML file
// run with: entering HTML file in a web browser
//   file:///home/student/hello_js.html
document.write ("Hello, World!") ;
```

```
<html>
<body>  <script type="text/javascript" src="hello_js.js"></script>   </body>
</html>
```

JavaScript's behaviors (ref)

- Compiled pgm size for Hello World: n/a

- OO Model: Yes, optional
- Semi-colon requirement as stmt terminator: Required

- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): Event Loop for asynchronous actions
- Program behavior on Divide by zero: Stopped by interpreter
- Garbage Collector (for objects): Yes
- Memory safe: Yes
- Thread safe: Yes
- DJ's Difficulty Index: 3 out of 10
- Language website: ecma-international.org/publications-and-standards/standards/ecma-262/

- Frameworks: NodeJS (js runtime env), jQuery, Web browsers, Angular, METEOR, Express, React
- Variable types: Number, String, Boolean, Object
- Behavior with unused variables (must use every variable defined?): No
- Can overrun variables: No
- Error detection mechanism: try/catch code block
- Program behavior on Invalid File I/O: Continue
- Pointer safe: Yes / n/a
- Variable safe (overwrite, wrap-around, buff overruns): Yes
- Goto statement: No, but keyword exists in language
- Release cycle: ECMA-script yearly. Node.js every 6 months

PHP - 1995 - interpreted (#5 of 12)

General-purpose scripting language geared toward web development

Server-side of websites

On Linux, can be run from the command line

Program does not stop on error; keeps running

```
<?php
// a comment - hello_php.php     #    another comment
// To run on Linux: php hello_php.php

echo "Hello, World!";
?>
```

PHP's behaviors (ref)

- Compiled pgm size for Hello World: n/a

- OO Model: Yes, optional
- Semi-colon requirement as stmt terminator: Not used

- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): With the "parallel" class that requires a build of PHP with ZTS (Zend Thread Safety)
- Program behavior on Divide by zero: **Continue running**
- Garbage Collector (for objects): Yes
- Memory safe: Yes
- Thread safe: Yes
- DJ's Difficulty Index: **2 out of 10**
- Language website: php.net

- Frameworks: Web servers, Laravel, CodeIgniter, CakePHP, Symfony, Zend
- Variable types: Number--Integer, Number-Float, String, Boolean, Object
- Behavior with unused variables (must use every variable defined?): No
- Can overrun variables: No
- Error detection mechanism: try/catch code block
- Program behavior on Invalid File I/O: **Continue**
- Pointer safe: Yes / n/a
- Variable safe (overwrite, wrap-around, buff overruns): Yes
- Goto statement: Yes
- Release cycle: Minor releases once per year around Nov/Dec. Point releases more often; perhaps one per month

Ruby - 1995 - Interpreted (#6 of 12)

Created from a desire for an object-oriented scripting language – not Perl or Python

Around 2005, popularity soared due to Ruby on Rails web framework written in Ruby

A focus on simplicity and productivity. Elegant syntax - natural to read; easy to write

Used primarily for Web development, Static website page generation, DevOps automation

Behind the scenes, everything is an object (like in Python)

Large standard library, especially for web stuff (YAML, JSON, XML, OpenSSL, etc)

On creating variables: start with lowercase creates Variable; uppercase creates Constant. Reassign Constant? Warns, continues

VARIETY of ways to perform terminal input/output but all of them are easy/straightforward

Ruby is **not** widely used to learn programming (first language) BUT it is free-form, quite forgiving, intuitive error messages

```
# a comment - hello_rb.rb     # To run on Linux: ruby hello_rb.rb
puts "Hello, World!“
print “Type in your first name: “
name = gets   # but normally we use gets.chomp to remove the carriage return char passed from the keyboard
print “Your name is “ + name + “\n”
print “Your name is #{name}”
apple = 5  ;  pear = “A pear”  ;  apple = apple + 1
if apple == 6 then puts “HaHa” ; pear = 2 ; bic = “awesome” ; end
```

Ruby

Ruby's behaviors (ref)

- Compiled pgm size for Hello World: n/a
- OO Model: Yes, optional
- Semi-colon requirement as stmt terminator: Optional. Line break is stmt terminator but is also optional. Sometimes we can have multiple stmts on same line.
- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): Native threads and fibers (lightweight)
- Program behavior on Divide by zero: Stopped by interpreter
- Garbage Collector (for objects): Yes
- Memory safe: Yes
- Thread safe: Yes
- DJ's Difficulty Index: **4 out of 10**
- Language website: ruby-lang.org
- Frameworks: Ruby on Rails
- Variable types: Number--Integer, Number—Float, String, Boolean
- Behavior with unused variables (must use every variable defined?): No
- Can overrun variables: No
- Error detection mechanism: Exception handling
- Program behavior on Invalid File I/O: Continue
- Pointer safe: Yes / n/a
- Variable safe (overwrite, wrap-around, buff overruns): Yes
- Goto statement: No
- Release cycle: 1+ times per year across 4 active release trains

Java - 1995 - compiled to Java byte-code; run on Java virtual machine (#7 of 12)

Designed to be safer than C, easier OO model than C++, and the same program can run on multiple OS platforms
Compiled to Java byte-code; run on Java virtual machine (JVM)
A concept of the JVM: multi-platform code; write once, run anywhere (limitations apply such as OS filename formats, window icons)
Java is Object-Oriented. Java language **reQUIRES** OO from the get-go. OO is not optional in Java
Java syntax is known to be verbose
Keyboard input is possible but can be lengthy. To avoid variable overruns, some peeps write routines that include backspacing
Java is used frequently for the back-end of websites, and for business process logic
Java was the official language for Android app development until Kotlin became the preferred language in 2018

```
/**************************************************************************
 *  Multi-line comment
 *************************************************************************/
// Single line comment
// compile with: javac hello_java.java
//    This compiles to filename: hello_java.class
// run with: java hello_java

public class hello_java {
   public static void main(String[] args) {
      int orange = 5;
      System.out.println("Hello, World!");
   }
}
```

Java's behaviors

(ref)

- Compiled pgm size for Hello World: **less than 1 kB (427 bytes)**
- OO Model: **Yes; mandatory**
- Semi-colon requirement as stmt terminator: Required
- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): Can start new threads
- Program behavior on Divide by zero: Stop / Fatal error
- Garbage Collector (for objects): Yes
- Memory safe: Yes
- Thread safe: Yes
- DJ's Difficulty Index: **7 out of 10**
- Language website: oracle.com/java
- Frameworks: Spring
- Variable types: Integers (diff sizes, signed/unsigned), Floats, Char, Byte, String, Boolean
- Behavior with unused variables (must use every variable defined?): No
- Can overrun variables: No
- Error detection mechanism: try/catch/finally block; try <resource> / catch block
- Program behavior on Invalid File I/O: Continue
- Pointer safe: Yes
- Variable safe (overwrite, wrap-around, buff overruns): Yes
- Goto statement: No, but keyword exists in language
- Release cycle: Every March and September

Kotlin - 2011 - compiled to Java byte-code; run on Java virtual machine (#8 of 12)

Kotlin was designed as an easier alternative to Java that runs on the JVM

Kotlin compilation speed is slow... ... Execution speed is comparable to Java

Became the official language for Android in 2018

JVM-related names to compile and run might be tricky for a beginning user

```
// a comment  - hello_kt.kt     /* another comment */

// Compile with: kotlinc hello_kt.kt
//   this ends as <name>.kt  (must end in .kt)
//   we compile to name: Hello_ktKt.class
// run with: java Hello_ktKt
//    this is H<name>Kt  (uppercase H, uppercase K, lowercase t)

fun main () {
   println ("Hello, World!")
}
```

Kotlin's behaviors

(ref)

- Compiled pgm size for Hello World: **less than 1 kB (645 bytes)**
- OO Model: Yes, optional
- Semi-colon requirement as stmt terminator: **Optional. ; is NOT usually required**
- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): Coroutines (lightweight threads)
- Program behavior on Divide by zero: Stop / Fatal error
- Garbage Collector (for objects): Yes
- Memory safe: Yes
- Thread safe: Yes
- DJ's Difficulty Index: **5 out of 10**
- Language website: kotlinlang.org
- Frameworks: Ktor, Kweb, Javalin, Spark , Spring Boot
- Variable types: Integers (diff sizes, signed/unsigned), Floats, Char, Byte, String, Boolean (like Java; on JVM)
- Behavior with unused variables (must use every variable defined?): No
- Can overrun variables: No
- Error detection mechanism: try/catch code block to detect error
- Program behavior on Invalid File I/O: Continue
- Pointer safe: Yes
- Variable safe (overwrite, wrap-around, buff overruns): Yes
- Goto statement: No
- Release cycle: Minor release about every 6 months. Point releases can be quite frequent

C# - 2000 - compiled (Microsoft Windows platform) (#9 of 12)

Somewhat of a Microsoft variant to Java

C# is currently quite popular as a language on the Microsoft Windows platform

Compiled to Intermediate Language (IL) in .exe file; run by Common Language Runtime with a Just-In-Time compiler

```
// a comment - hello_cs.cs
// compile with: csc hello_cs.cs  --OR-- MS-Visual Studio
namespace HelloWorld
{
   class Hello {
      static void Main (string[] args)
      {
         System.Console.WriteLine ("Hello, World!");
      }
   }
}
```

C#'s behaviors (ref)

- Compiled pgm size for Hello World: **4096 bytes standalone (Windows)**
- OO Model: Yes; optional but highly pushed
- Semi-colon requirement as stmt terminator: Required

- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): Thread and Task object classes
- Program behavior on Divide by zero: Stop / Fatal error
- Garbage Collector (for objects): Yes
- Memory safe: Yes
- Thread safe: Yes
- DJ's Difficulty Index: **7 out of 10**
- Language website: learn.microsoft.com/en-us/dotnet/csharp/

- Frameworks: ASP.Net
- Variable types: Integers (diff sizes, signed/unsigned), Floats, Char, String, Boolean
- Behavior with unused variables (must use every variable defined?): No
- Can overrun variables: No
- Error detection mechanism: try/catch code block to detect error
- Program behavior on Invalid File I/O: .Net throws exception
- Pointer safe: Yes
- Variable safe (overwrite, wrap-around, buff overruns): No
- Goto statement: Yes
- Release cycle: Visual Studio is released about twice per month

Go - 2009 - compiled (#10 of 12)

Developed by Google; used at Google -- their much-preferred alternative to C++
Core objectives are code safety and simplicity; minimalist approach. Minimum # of keywords. Only one way to perform a loop in Go
Requires program to be "production-ready" code
Opening brace MUST be on same line as function name
Compiler does not handle some arrangements of statements / blocks over multiple lines
Aborts compilation on Unused variables
Commenting out a section of code can be a “bear” because non-referenced variables abort the compilation
Native multitasking (Goroutines)
A Go program can require a lengthy list of import statements
Other languages have several loop structures (e.g. for, do/until, do/while), Go implements all or these with the “for” loop
Keyboard input is possible in Go, but this requires several “import” statements and a few lines of code

```
package main

// a comment - hello_go.go     /* another comment - hello_go.go  */
// compile with: go build hello_go.go

 import "fmt"
func main () {
 fmt.Println ("Hello, World!")

}
```

Go's behaviors

(ref)

- Compiled pgm size for Hello World: **1.2 MB includes run-time by default**
- OO Model: **OO-ish; not full OO model**
- Semi-colon requirement as stmt terminator: Optional

- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): **Native with Go-routines**
- Program behavior on Divide by zero: Stop / Fatal error
- Garbage Collector (for objects): Yes
- Memory safe: Yes
- Thread safe: Yes
- DJ's Difficulty Index: **5 out of 10**
- Language website: go.dev

- Frameworks: Goji, Gin, Beego, Echo, Buffalo, Gorilla
- Variable types: Integers (diff sizes, signed/unsigned), Floats, String, Boolean
- Behavior with unused variables (must use every variable defined?): **Yes**
- Can overrun variables: No
- Error detection mechanism: **functions return err code in addition to value result**
- Program behavior on Invalid File I/O: Continue
- Pointer safe: Yes
- Variable safe (overwrite, wrap-around, buff overruns): No
- Goto statement: Yes
- Release cycle: Every 6 months in February and August

Rust - 2010 - compiled (#11 of 12)

Developed by Mozilla Inc. Used for Firefox and other projects

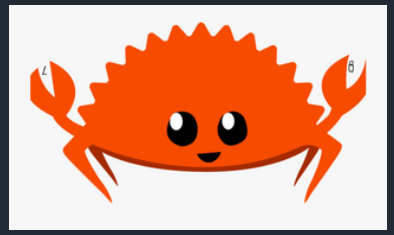
Chosen as alternative to C for use in the Linux kernel

Warns when variables are unused; opening brace can be on subsequent line

Feature-rich; significant learning curve

```
// a comment  - hello_rs.rs
//  Compile with rustc hello_rs.rs

fn main ()
{
   println! ("Hello, World!");
}
```



Rust's behaviors (ref)

- Compiled pgm size for Hello World: **381 kB**
- OO Model: OO-ish; not full OO model https://doc.rust-lang.org/book/ch17-00-oop.html
- Semi-colon requirement as stmt terminator: Required
- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): Can start new threads via function calls
- Program behavior on Divide by zero: Stop / Fatal error
- Garbage Collector (for objects): No*. **At compile time perform drop() at end of variable lifetime. Other options in std lib and in crates**
- Memory safe: Yes
- Thread safe: Yes
- DJ's Difficulty Index: **8 out of 10**
- Language website: rust-lang.org
- Frameworks: Actix, Rocket, Axum, warp
- Variable types: Integers (diff sizes, signed/unsigned), Floats, Char, Boolean, String object
- Behavior with unused variables (must use every variable defined?): No
- Can overrun variables: No
- Error detection mechanism: Result<T, E> for recoverable error; panic for unrecoverable error
- Program behavior on Invalid File I/O: Continue
- Pointer safe: Yes
- Variable safe (overwrite, wrap-around, buff overruns): No
- Goto statement: No
- Release cycle: Stable release every 6 weeks

Swift - 2014 - compiled (#12 of 12)

Developed as a replacement for Apple's older Objective C. Swift is used heavily on Apple platforms

On Apple platforms, Swift uses Objective C runtime library (interoperability). Different runtime library on non-Apple platforms

Extensive Object model and features

C-style language but several C-style constructs are changed/removed to reduced errors (see Wikipedia)

On creating variables… *var* keyword (optional) creates Variable but *let* keyword creates a Constant

Compiler nicely parses multi-line statements / code blocks with braces on different lines

Swift supports "Optionals" where a variable (var) might be nil and not have a value. Some functions REQUIRE a value and might need to be forced, e.g. with an (!) exclamation point. This can complicate things when you need a "var" and not a "let" (constant)

A code anomaly can generate several error or warning messages; not always identifying the actual problem or the parameter that is causing the problem. 'Good times in knowing where to place / NOT place ! or omitting a : before an optional variable type

```
// a comment - hello_swift.swift
// compile with:  swiftc hello_swift.swift -o helloswift
print ("Hello, World!")
var otherstring: String = "All kinds of stuff"
var response: String! = " "
print ("Enter a string: ", terminator: "" )
response = readLine ()
print (response!)   ;   print (otherstring)
```

Swift's behaviors (ref)

- Compiled pgm size for Hello World: 21 kB

- OO Model: Yes
- Semi-colon requirement as stmt terminator: **Optional; can be used to place several stmts on one line**

- Safer input into String objects; or routines that limit input: Yes
- Concurrency (Single-threaded vs concurrent processing): **Built-in asynchronous and parallel code. Introduced in version 5.3**
- Program behavior on Divide by zero: Stop / Fatal error
- Garbage Collector (for objects): Yes (Automatic Ref Count)
- Memory safe: Yes
- Thread safe: Yes
- DJ's Difficulty Index: **7 out of 10**
- Language website: swift.org

- Frameworks: Cocoa, Cocoa Touch
- Variable types: Integers (diff sizes, signed/unsigned), Floats, Character, String, Object, Boolean
- Behavior with unused variables (must use every variable defined?): No (Swift prints a warning)
- Can overrun variables: No
- Error detection mechanism: **do/catch code block; defer to ensure cleanup is run**
- Program behavior on Invalid File I/O: Continue
- Pointer safe: Yes, generally (Ptrs are not exposed, by default)
- Variable safe (overwrite, wrap-around, buff overruns): Yes
- Goto statement: No
- Release cycle: Minor releases about twice a year; Point releases between them

“Hello World!” in Assembly (Linux)

```
; a comment - hello_asm.asm
; To Assemble and Link 32-bit code in Linux with NASM:
;    nasm -f elf hello_asm.asm     (add -g for debug)
;    ld -m elf_i386 hello_asm.o -o hello_asm
; To Assemble and Link 64-bit code in Linux with NASM:
;     nasm -f elf64 hello_asm.asm  (add -g for debug)
;     ld  hello_asm.o -o hello_asm

section .data
      helloString db "Hello, world!",10,0          ; String, LF, ASCIIZ
      len1 equ $ - helloString                     ; length of string (14)

section .text
      global  _start
_start:

      mov       edx,len1                           ; Write String to Stdout w Syscall
      mov       ecx,helloString                    ; load string length
                                                   ; load pointer to the string to write
      mov       ebx,1                              ; load file handle (1 is stdout)
      mov       eax,4                              ; load system call number (sys_write)
      int       0x80                               ; invoke Interrupt to call OS

                                                   ;Exit Program
      mov       ebx,0                              ; load exit code (0 = normal completion)
      mov       eax,1                              ; load system call number (sys_exit)
      int       0x80                               ; invoke Interrupt to call OS

                                                   ; note 32-bit hello_asm executable is 9076 bytes   64-bit is 9424 bytes   Much of this is Linux exe format
```

```
C:\WINDOWS\SYSTEM32>debug
-a
0CBB:0100 mov ax,0
0CBB:0103 mov ax,cx
0CBB:0105 out 70,al
0CBB:0107 mov ax,0
0CBB:010A out 71,al
0CBB:010C inc cx
0CBB:010D cmp cx,100
0CBB:0111 jb 103
0CBB:0113 int 20
0CBB:0115
```

Languages – Compile and Execution speeds

Language	Compile speed	Execution speed
C	- Very fast	the FASTEST
C++	- Fast	Fast
Python	- n/a	Moderately Fast (but slower than PHP)
JavaScript	- n/a	Slow
PHP	- n/a	Fast (but slower than C, C++, Go, Rust. Very fast for an interpreter)
Ruby	- n/a	Moderate (*)
Java	- Moderate	Moderate
C#	- Moderate	Moderate
Kotlin	- Slow	Moderate (equivalent to Java)
Go	- Moderate	Fast (slightly slower than C++, Rust)
Rust	- Moderate	Fast (similar to C++, sometimes faster)
Swift	- Moderate	Fast (slightly slower than PHP)

(*) The Moderate execution speed languages are similar, given the natural variances in the test trials and their tendencies of time spent in user and system execution. They are also similar in wall-clock times

G-P L vs Systems Language vs Applications Language

G-P L - generic features that allows the same code on different platforms

Systems Language – to create systems software: OS, compilers, interpreters, assemblers, utilities

Some Systems programming languages…

- Legacy systems languages:
  - Assembly, PL/1, Pascal
- Current systems languages:
  - C, C++, Rust, Go, Swift

Some current Applications languages:

- C++, Rust, Go, Swift, C# (compiled), Java, Kotlin, Python, JavaScript, PHP, Ruby, Perl (interpreted)

Languages Used to write Compilers, Interpreters

Language	Written in	Used for
C	C and C++; orig New B	OSes, Languages, glibc, embedded systems
C++	C and C++; orig C	OSes, Games
Perl	C	"Glue"/integration language
Python	C	applications, middleware, integration, ML/AI
JavaScript	C	website client-side in browser
PHP	C	website server-side
Ruby	C	
Java	Java (compiler), C (JRE); orig C and some C++ libs	applications, middleware, Android apps
C#	C#; orig C and C++	MS-Windows applications
Kotlin	Java and Kotlin; orig Java	applications, Android apps
Go	88% Go, 6% Assembly; orig was C	applications
Rust	**Rust**; orig OCaml	applications, Linux Kernel (OS)
Swift	50% C++, 5% C, 41% Swift (std lib)	iPhone/iPad apps, MacOS applications

Languages Used to write OSes

OS	Written in
Android	- mostly C, then a layer of Java, apps in Java and Kotlin
iOS	- C, C++, Objective C, Swift, Assembly
Linux	- C, small amount of Assembly, now adding some Rust
MacOS	- C, C++, Objective C, Swift, Assembly
Windows	- C, C++, C#, Assembly

(Before C, most OSes and languages were written in Assembly)

Brief considerations in choosing your language

- For certain tasks it is easy: JavaScript/frameworks for browser; PHP, Python, Java, Ruby for back-end
- For other tasks the choice is more nuanced
- Whether we **can** use or **want** to use Interpreted code, Compiled binary, or Compiled bytecode
- 3 Factors that Help in choosing language(s) to use:
  - Platform, OS
  - Languages used by a certain type of program or industry
  - Preference of language style, vulnerabilities, features
- D.J.'s Picks:
  - ✓ (beginner) PHP from command line or Ruby; Kotlin or Python as a second or third language
  - ✓ (then) C
  - ✓ (followed by) Go or Rust
- Some people say C needs to go away or C is going away...
  - D.J. believes C will be around for a long time: C is the base of OSes, glibc, and many languages

How does C-I-A manifest itself in code?

Let's review::

- Confidentiality - Keep private information private (Disclosure)
- Integrity - Data/program/function/Design is not changed (Alteration)
- Availability - Data/program/system can be accessed and operated as designed (Destruction)

CIA inverse DAD

There are tensions among the C-I-A attributes

- Example: A program keeps running (availability) with an I/O error or divide by zero but the results are incomplete or less accurate (integrity)
- Example: A program is coded well to stop running on questionable input or program errors (integrity) but a minor issue prevents system processing from occurring (availability)
- Example: The requirement of encryption (confidentiality) possesses a risk of changing data (integrity) or making data unavailable (availability)

Types of S/W Vulnerabilities in Languages (1)

Insufficient Input Validation

- Overwriting a variable because Input String is too long
- Input is appended to SQL/DB request and crafted field input modifies query
- Input is appended to OS request and crafted field input modifies query
  - Result: Running unintended queries/commands. Directory traversal is a targeted example of this

Too large a value (e.g. number) can overflow into adjacent variable (integer overflow)
Too large/small/different a value can alter program logic in unintended/unchecked way

Unanticipated/unchecked error conditions can alter program logic in unintended/unchecked way
Unanticipated program stop OR continuation from an error conditions can alter program logic in unintended/unchecked way

Developer Misunderstands the behavior of statements, functions, frameworks, anything else... ... (logic, program errors)

- Example: In C / C++ / Python, a logic error occurs if we use the exit statement without parentheses
- Example: Python round function can truncate (round down) at .5 per the function's Design

Addressing invalid memory - Pointer de-referencing to a null value
Resource exhaustion / memory leaks

Types of S/W Vulnerabilities in Languages (2)

Time of Check / Time of Use (TOC/TOU)

Example: At DC30 we saw the file substitution vulnerability with the Apple platform Zoom client Installer

Race conditions

Different or upgraded compiler/interpreter versions

Bugs in Assemblers, compilers, interpreters, linkers, Language libs, run-time

Examples:

- Mismatch between NASM assembler and linker prevented 64-bit source code from displaying in gdb (GNU Debugger)
- Assembly instruction to move 1 to rax 64-bit register (mov rax, 1). Assembler inserts instruction to move 1 to eax 32-bit register: the lower half of rax
- Processors have bugs. OSes code around them (Pentium FDIV, Meltdown, Spectre, Zenbleed, Speculative Store Bypass)

Bugs, vulnerabilities in libraries that you acquire; and in Frameworks

Example: Log4j, VM2 NodeJS JavaScript sandbox library

Intentional or Malicious changes by... ... Someone

Example: SolarWinds breach

Recommendations from Observations in Vulnerability Databases

- I Recommend good input validation, and edge case consideration during CODING and testing
- Code and test for 7 edge cases:
  - Exact/max value (e.g. 25 characters in 25 length field)
  - Zero length entry
  - One character
  - Length +/- 1
  - and Length +/- 2
- During application design and program design, identify the size/range limits of variables
- For functions/subroutines, add comments to indicate the size/range constraints of variables
- Until a developer is **extremely** familiar with a language, test every logic branch
- Languages evolve. Changes in Statements/Keywords can break a program in the future. Review language release notes
  - Watch out for: Use of current keywords/functions, Future keywords
- Frameworks can introduce vulnerabilities. IDEs can change/break an interpreted program's behavior
- Valgrind and profilers are great testing tools
- Become familiar with using debuggers: gdb (GNU DeBugger), pdb (Python DeBugger), Visual Studio debugger
- For Linux, AWS reduces Pointer abuse with their Gravaton processors. Compilers produce code to protect pointer operations

From a tough, yet beloved, Systems Analyst lady who became a Dev Manager and Department Director: I should be able to put my BUTT on the keyboard and the program not crash… … …

Counting things… Variations in the Ways Languages Process Loops

J	O	H	N	S	O	N	-	S	M	I	T	H	\0	???	???
4012	4013	4014	4015	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

PTR

Mem Addresses
Index
Subscript

C, C++, C#, Java, JavaScript, Go, PHP
for (x = 1; x < 11; x++) print the number ; // counts to 10 D.J. prefers to use x <= 10 but this can generate more Assembly instructions
Python
for x in range (1, 10): # counts to 9
Ruby
for x in 1 .. 10; print the number ; end # counts to 10
for x in 1 … 10; print the number ; end # counts to 9
Kotlin
for (x in 1 .. 10) print the number // counts to 10
Rust
for x in 1 .. 10 { print the number } // counts to 9
for x in 1 ..= 10 { print the number } // counts to 10
Swift
for x in 1 … 10 { print the number } // counts to 10
for x in 1 ..< 10 { print the number } // counts to 9

There are numerous vulnerabilities in the CVE database that are caused by “counting” errors … …

Traditional Resources for Vulnerability Prevention

CVE Database by Mitre Corp - cve.mitre.org --transitioning-to--> www.cve.org

Search - cve.mitre.org/cve/search_cve_list.html

NIST National Vulnerability Database

nvd.nist.gov/vuln/detail/CVE-yyyy-nnnnn

Center for Internet Security (CIS)

CIS Controls, CIS Benchmarks, Hardened Images

https://www.cisecurity.org/

OWASP

Cheatsheat Series – Numerous languages, platforms, implementations, technologies, vulnerabilities

https://cheatsheetseries.owasp.org/IndexProactiveControls.html

API Security Top 10 - 2023

https://owasp.org/www-project-api-security/

Conclusion

Sooo, which language SHOULD we use for less vulnerable code??

No Clear Winner. But … …

Newer languages (e.g. Kotlin, Swift, Go, Rust) provide some features that can reduce bugs in some cases

- Objects / OO-ish behavior (they don't overrun variables easily); Easier object handling (fewer mistakes); Efficient garbage collection (reduces resource exhaustion); Less dependence on pointers (reduces data errors, resource exhaustion, program crashes)

Interpreters offer these benefits, good input protection, good protection of variables but interpreters are less suited for systems work and high performance applications (like video games)

ALL languages have some desirable use-cases (even C for OSes and languages)

For ANY language we still need to exercise due care with the ideas that are presented here today

We have outlined a few ways to foster less buggy/vulnerable code

We have provided Examples to start programming in 12 languages

Remember: There is ALWAYS another … B ug … …

Thank you!

D.J. Davis (ZeroRingDefender)
ZeroRingD@gmail.com
Twitter/X: @ZeroRingD

https://github.com/ipv3/DC31-BIC/

< Slide deck - Program examples - Set up, back up, restore Linux - GDB commands />