classmate
Date 13/8/2019
Page 1

PL ATTACK STRATEGY

attack1: REPL

no need for main function

```
#include <stdio.h>
printf ("hello world");
// how to return value from main? (exit code)
```

attack2: simpler import system (no .h, .lib)

```
import stdio
printf ("hello world");
```

attack3: no need for preprocessor!!!

wait...

how to support high order functions in c? /c++?

4 hrs. functions can be stack-based like in C, or register based like it is for software interrupts in DOS (QB). (Ralf brown interrupt list)

strings can be graphs internally

using a small hashcode for fast equality ~~search~~ sort. check.

array of numbers. may also be graphs internally.

- missing out the base condition in recursive function.

```
int factorial(int n) {
  return n * factorial(n-1);
}
```

- using character constants instead of string literals, and vice versa.

- floating fpoint literals are of type double by default.

- forgetting to free memory.

```
getline(&line, &size, stdin);
return 0;
```

-

- adding a semicolon to #define

- be careful with semicolons (didnt sleep for 2 days because of semicolon)

- mistakenly writing = instead of == when comparing (croroots killing homans)

- copying too much (buffer overflow)

- macros are simple string replacements.

- double free call (free, and not set to null)

Copy constructor
assignment operator.
C needs better standard library.

compiler constructors, destructors, assignment operators
poor tools for safety, strings & others
no RAII

11. most common pitfalls in C programming language

- mixing signed & unsigned integers in arithmetic operations

```
signed int a = -1
unsigned int b = 1000
print ( b > a )     // false!
```

- overstepping array boundaries

- missing out the base condition in recursive function

```
int myArray[] = {1, 2, 3, 4, 5};
for ( x = 1; x <= 5; x++) {
    print ( myArray[x] )
}
// 2, 3, 4, 5, garbage
```

GENERICS IN JAVA

```
Map<String, List<String>> someMap;
```

extra typing
lots of cruft.

```
List<String> someList = (List<String>) someMap.get("someKey");
```

```
public class MyClass {
  public T getStuff() {
    return new T();
  }
}
```

- type erasure
- no support for primitive types
- incompatibility with annotations
- incompatibility with arrays
- weird wildcard syntax and behaviour.
- inconsistent support.

classmate
Date 12.8.2019
Page 1

WHY I LOVE JAVA

- memory hogging garbage collector (selfish algorithm, forces you to forget about freeing).
- breaking of module boundaries with resource handling "allocate any resource and forget" (automatic destructor calling)
- who should manage / maintain a shared resource, (good OO design - clear responsibility separation).
- GC is efficient for small allocations, but not good for larger allocations.
-

KEYWORDS IN C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	static	void
default	goto	sizeof	volatile
do	if	signed	while

ⓘ can keywords have expanded meanings
ⓘ can we provide suggestions to compiler
ⓘ can we use html through literate programming.
ⓘ can we translate to javascript (from).
ⓘ can we compile to web assembly. (java, jit).

→ java, haskell, racket, go, julia, js

- high order functions
- 'match' expression (replacing 'switch')
- import cycle detection
- 'crypto/md5', 'crypto/sha256', 'crypto/sha512' mods.
- 'os.executable()' a cross-platform function that returns full path to current executable.
- ~/.vlang, VROOT removed entirely. installation lot cleaner.
- v can now be packaged for all linux distros.
- arch linux package.
- string(bytes buffer, len), string(bytes_array) casts.
- multiple defers.
- 'key in map' syntax (replacing map.exists(key)).

v0.1.17

- 'vweb' module for developing web apps in V.
- vtalk, open source v forum software (more!)
- generics (very limited right now, but they will be gradually improved).
- comptime codegen (foo.$method() where method is a string)
- @ for escaping keywords (eg. struct Foo { @type string })
- windows unicode fixes (v can work with non-ASCII path on windows)
- fix mutable args bugs + dont alloca primitive args to be modified.
- declaring a mutable variable, never modifying it results in compilation error.
- interactive debugging support
- 'sync' module for windows.
- '#!' support for unix systems (vscripts)
- lots of visual studio fixes.
- 'crypto.aes' & 'crypto.rc4' modules.

alexa. web ui html css

→ JS

classmate
Date 12.8.2019
Page 3

- `<<` can now append arrays (`numbers << [1 2 3]`)
- lots of windows fixes
- lots of REPL improvements (>> 2+3 ; no println)
- website easily translatable, now partially available in several languages.

V0.1.15

- `gg` module windows support, v tetris runs on windows.
- `glad`, `cJSON` now compiled only once, ...
- v.c cleaned up and minimized. (~16k => ~10k loc).
- `type` aliases can now have methods.
- const overflow check during compilation (byte(1000) X)
- freebsd, openbsd, netbsd, dragonfly support.
- hot code reloading now works with graphical applications bounce.v
- VROOT removed, installation process now much simpler.
- `defer` statement
- map.v rewritten. its now much faster.
- `for key, val in map` syntax.
- `flag` module for parsing ~~cmd~~ command line arguments
- `zip` module
- `crypto/sha1` module
- submodules and module aliases (import encoding.base64 as b64)

V0.1.16

- V can now be used with visual studio
- hot code reloading now works with graphical applications graph.v, bounce.v
- compile time memory management for arrays.

- windows :

v 0.1.10

- windows support via minGW-w64 . pre-built windows bin.
- file structure simplified. all vlib in vlib/.
- mut var := val fixed

v 0.1.11

- cross compilation for windows
- lots of windows fixes
- socket.v
- maps fixed

v 0.1.12

- now windows compile works
- `os` module now uses optionals in functions -> `File`
- println optimized - no longer allocation needed.
- lots of vfmt fixes.
- new `strings` module
- lots of other fixes & improvements

v 0.1.13

- new enum sytax (token == .name) enum no longer global.
- submodules (import encoding.base64)
- hot code reloading
- special err variable for getting error values
- complex numbers

easy string forming? string within string?

KEY FEATURES OF V

- simplicity - can be learned in ½ hr, less if go
- fast compilation - 100k loc ⟶ 1-2M loc
- easy to develop - self compile < 1s
- performance - within 5% of C
- safety - no null, no globals, no undefined behaviour, immut. def.
- C to V translation
- hot code reloading
- powerful ui & graphics libraries
- easy cross compilation
- REPL (interpreter) (jit) (vm)

RELEASES

V0.1.7

- all C code in vcompiler and vlib replaced with V.
- # syntax for embedding C code removed
- exported functions now need to be marked with 'pub', all public vlib functions updated.
- cleaner bytes to string conversion tos2(C...) ⇒ string(bytes)
- 3 more examples next to 'hello world'.
- "bugs look dangerous", dave

V0.1.8

- single file programs without fn main now work as expected.
- repl fixed, now supports imports, consts, func. defs.

PROBLEMS FACED

- asciiz string / text → custom spec "... \0 ..." ok
- bracket confusion with {[( )]} → any ((2+3)x8)
- operator overloading replacement. → "has" + "soot"
- easy custom datatypes — string(1) / (2) / (4) (templates)
- small runtime environment
- running it on a VM (LLVM).
- from popl — literate programming
- no datatype always specify (generics)
- pass by value vs pass by reference
- lambda functions and pure functions
- constant variables vs modifiable variables
- less loop types, conditionals.
- only 1 way to write library (basic).
- multiprocessing easy.iq
- event based programming (promise) future?
- debugging and performance measurement.