Optimising for humans

Nim meta-programming for 0-cost abstraction on microcontrollers

PMunch – peterme.net
Peter Munch-Ellingsen, M.Sc

Programming languages

```
print:
  movw  $BOOT_SEGMENT, %ax
  movw  %ax, %ds
print_loop:
  lodsb
  cmpb  $0,%al
  je    print_done
  movb  $14, %ah
  movl  $0x0002, %ebx
  int   $0x10
  jmp   print_loop
print_done:
  retw
```

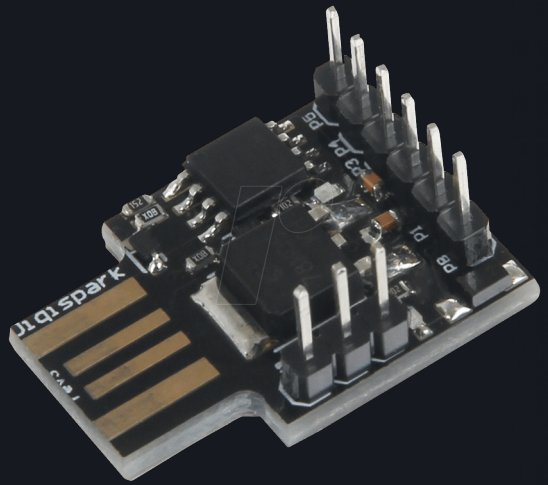
```
int main(int argc, char **argv)
{
    progname = argv[0];
    options.vm = 0;
    options.extended = 0;
    while (argc > 1)
    {
        char *option = &argv[1][2];

        if (strcmp (option, "vm") == 0) {
            options.vm = 1;
        } else if (strcmp (option, "tm") == 0) {
            options.extended = 1;
        }
        argc--;
    }
}
```

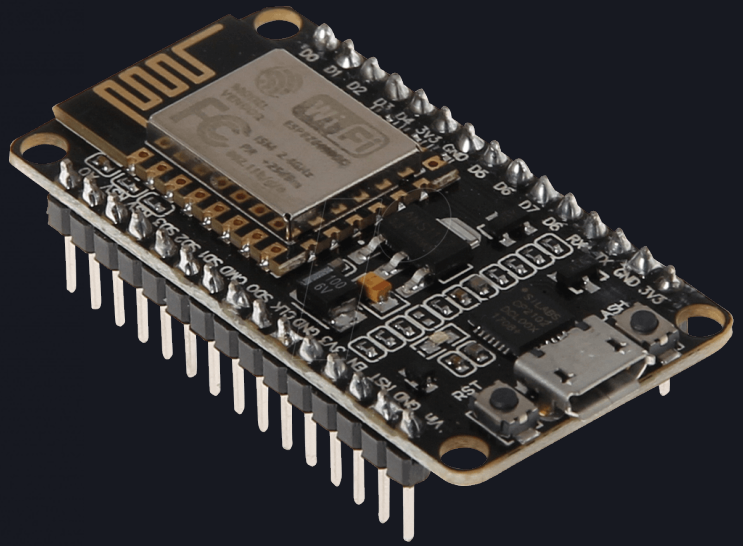
Programming languages

```
(defn send-value
     "Sends a key value pair to a server"
     [host port k value]
     (with-open [sock (Socket. host port)]
         (send-msg sock (str "PUT "
             k ":" value \newline))
         (receive-msg sock)))
```

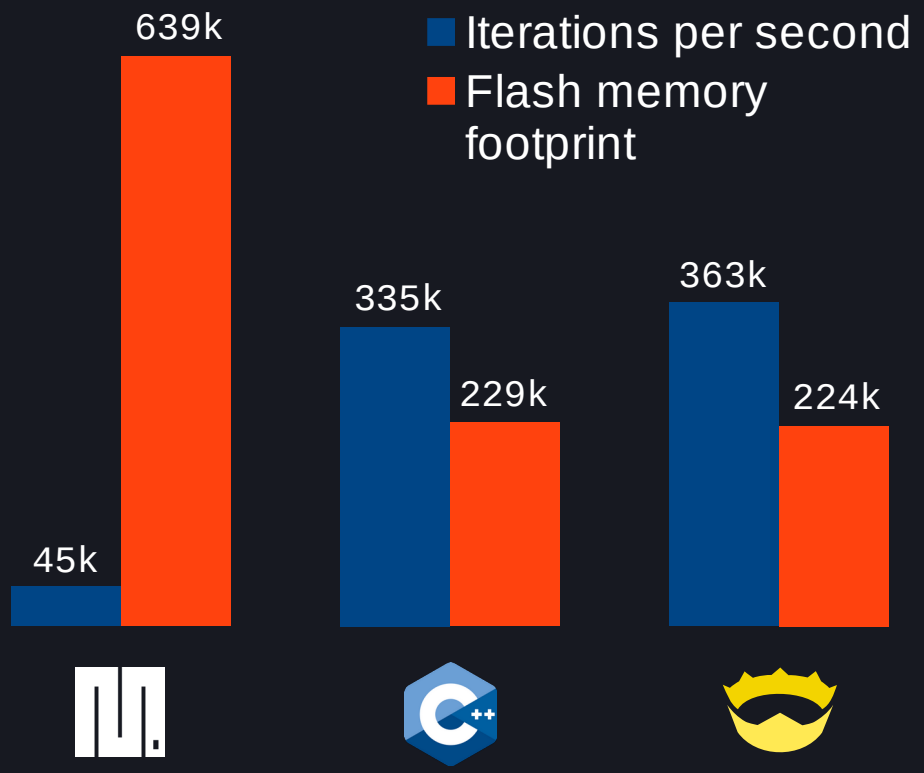
```
var
  sum = 0
  count = 0

for line in stdin.lines:
  sum += line.len
  count += 1

echo("Average line length: ",
     if count > 0: sum / count else: 0)
```

Small targets





Performance comparison



What is Nim?

- Compiled
- Statically typed
- Speed of C,
  ease of Python,
  flexibility of Perl
- Controllable GC

```
# Compute average line length
# From nim-lang.org
var
  sum = 0
  count = 0

for line in stdin.lines:
  sum += line.len
  count += 1

echo("Average line length: ",
     if count > 0: sum / count else: 0)
```

Nims killer feature

```nim
import macros, strutils

macro toLookupTable(data: static[string]): untyped =
  result = newTree(nnkBracket)
  for w in data.split(';'):
    result.add newLit(w)

const
  data = "mov;btc;cli;xor"
  opcodes = toLookupTable(data)

for o in opcodes:
  echo o
```

AST based macros

```
var x = 100
if x > 10:
  echo x.float
```

```
StmtList
  VarSection
    IdentDefs
      Ident "x"
      Empty
      IntLit 100
  IfStmt
    ElifBranch
      Infix
        Ident ">"
        Ident "x"
        IntLit 10
    StmtList
      Command
        Ident "echo"
        DotExpr
          Ident "x"
          Ident "float"
```

Making a game in Nim

Leveraging compile-time powers

```
type
  Sprite[count, w, h: static[int]] = distinct array[count, uint8]

template loadSprite*(name, file: untyped): untyped =
  sprite(name, loadBMP(file))

macro sprite*(name: untyped, x: static[string]): untyped =
  [...]
  result = quote do:
    let
      `name` {.codegenDecl: "const $# PROGMEM $#".} =
       Sprite[`spriteLen`, `cols`, `rows`](`spriteData`)
```

Leveraging compile-time powers

```
const spritePath {.strdefine.}: string = "sprites/"

loadSprite(leg1, spritePath & "leg_frame_1.bmp")
loadSprite(body, spritePath & "body.bmp")
loadSprite(head, spritePath & "head.bmp")

proc drawPlayer() =
  calculatePlayerBounds()
  drawSprite(x, y-hy, head)
  drawSprite(x+2, y+7-by, body)
  drawSprite(x+1, y+11-by, frame)
```

Leveraging compile-time powers

```
template drawBitmap*[count, w, h: static[int]]
                   (x, y: int16, bitmap, mask: Sprite[count, w, h]) =
  drawBitmap(x, y,
             cast[ptr uint8](bitmap.unsafeAddr),
             cast[ptr uint8](mask.unsafeAddr),
             w, h)

template drawBitmap*[count, w, h: static[int]]
                   (x, y: int16, bitmap: Sprite[count, w, h]) =
  drawBitmap(x, y,
             cast[ptr uint8](bitmap.unsafeAddr),
             nil,
             w, h)
```

Leveraging compile-time powers

```
type
  LevelData[count: static[int]] = distinct array[count, uint8]
  PositionData[count, width: static[int]] = distinct array[count, uint16]

proc `[]`*(data: LevelData, idx: uint32): uint8 =
  pgmReadByte(cast[ptr uint8](cast[int](data.unsafeAddr) + idx.int))

proc `[]`*(data: PositionData, idx: SomeInteger): uint16 =
  pgmReadWord(cast[ptr uint8](cast[int](data.unsafeAddr) + (idx * 2).int))

macro loadPositions*(name: untyped, levelString: static[string]): untyped =
  [...]
    result = quote do:
    let `name` {.codegenDecl: "const PROGMEM $# $#".} =
      PositionData[`count`, `w`](`levelData`)
```

Fixed point arithmetic

```
defFixedPoint(Speed, int8, 4)
defFixedPoint(Position, int16, 4)

type
  Particle = object
    age: uint8
    x, y: Position
    xs, ys: Speed

[...]
for particle in particles:
  particle.x += particle.xs +
    (if scene != GameOver: toSpeed(1) else: toSpeed(0))
  particle.y += particle.ys
  particle.ys += toSpeed(0.1)
```

Zero-cost abstractions

```
proc loop() {.exportc: "loop".} =
  serialPrintLn("Loop")
  let startTime = millis()
  # Using zero-functional
  serialPrintLn $(zip(a, b) -->
                  map(f(it[0], it[1])).
                  filter((it mod 4'u) > 1'u).
                  map(it * 2'u8).
                  all(it > 4'u))
  # VS. Using the system sequtils
  serialPrintLn $(zip(a, b).
                  mapIt(f(it[0], it[1])).
                  filterIt(it mod 4'u > 1'u).
                  mapIt(it * 2'u8).
                  allIt(it > 4'u))
  serialPrint("Solved in: ".cstring)
  serialPrintLn($(millis() - startTime))
```

Optimising for humans

Nim meta-programming for 0-cost abstraction on microcontrollers

PMunch – peterme.net
Peter Munch-Ellingsen, M.Sc