Languages and Compilers
Syntax in Practice

Adam Sampson
School of Design and Informatics
Abertay University

Why syntax matters

- Language designers tend to concentrate on getting the **semantics** of the language right...
- … but **syntax** provides the interface between the programmer's brain and the compiler – and programmers get very attached to syntax
- We've looked at **syntax analysis** – but we haven't really talked about what syntactic oddities we might actually see in programming languages
  - as most books/courses don't
  - and there's much less research about syntax

Why *design* matters

- When designing a programming language, do we want it to be...

- … easy to read?
- ... easy to write?
- … easy to learn?
- … easy to parse/compile?
- … concise (or expressive)?
- … difficult to make mistakes in?

FORTH: nearly no syntax

```
: checkmem  ( ad n --- )
   0
   DO
      >R
      T{ R@ C@ -> R> I 1+ SWAP >R }T
      R> 1+
   LOOP
   DROP
;

: TESTING   \ ( -- ) TALKING COMMENT.
  SOURCE VERBOSE @
   IF DUP >R TYPE CR R> >IN !
   ELSE >IN ! DROP [CHAR] * EMIT
   THEN ;
```

ColorForth: even less syntax

```
IDE hard disk driver
bsy 1f7 p@ 80 and if bsy ; then ;
rdy 1f7 p@ 8 and if 1f0 a! 256 ; then rdy ;
sector 1f3 a! swap p!+ /8 p!+ /8 p!+ /8 e0
  or p!+ drop p!+ drop 4 * ;
read 20 sector 256 for rdy insw next drop ;
write bsy 30 sector 256 for rdy outsw next
  drop ;
```

Fortran IV (1961)

```
C       Irvin Levy (Gordon College) Linear Regression Package
C
        DIMENSION X(52), Y(2,50), LITERL(2)
        DOUBLE PRECISION S1,S2,S3,S4,S5,T,S,B,D,R,E1,E2,BBAR
        WRITE (5,10)
 10     FORMAT('0',1X,'*   *   *  LINEAR REGRESSION ANALYSIS  *   *   *',//)
        WRITE (5,20)
 20     FORMAT(1X,'HOW MANY PAIRS TO BE ANALYZED?'$)
        READ (5,*) N
        IF (N.GT.50) GOTO 70
        WRITE (5,30)
 30     FORMAT(//1X,'Enter one pair at a time')
        WRITE (5,40)
 40     FORMAT(1X,'and separate X from Y with a comma.'//)
        WRITE (5,50)
 50     FORMAT(1X,'Enter pair number one : '$)
        READ (5,*) X(1), Y(1,1)
                DO 60 I=2,N
                WRITE (5,55) I
 55             FORMAT(1X,'Enter pair number',I3,' : '$)
                READ (5,*) X(I), Y(1,I)
 60             CONTINUE
        GOTO 90
 70     WRITE (5,80)
 80     FORMAT(1X,'At present this program can only handle 50 data pairs.')
        STOP
 90     WRITE (5,100)
```

Lisp: program as data structure

```
(defun org-datetree-find-month-create (year month)
  "Find the datetree for YEAR and MONTH or create it."
  (org-narrow-to-subtree)
  (let ((re (format "^\\*+[ \t]+%d-\\([01][0-9]\\) \\w+$"
                    year))
   match)
    (goto-char (point-min))
    (while (and (setq match (re-search-forward re nil t))
      (goto-char (match-beginning 1))
      (< (string-to-number (match-string 1)) month)))
    (cond
     ((not match)
      (goto-char (point-max))
      (or (bolp) (newline))
      (org-datetree-insert-line year month))
     ((= (string-to-number (match-string 1)) month)
      (goto-char (point-at-bol)))
     (t
      (beginning-of-line 1)
      (org-datetree-insert-line year month)))))
```

ABC: significant whitespace

```
HOW TO QUEENS n:
    DISPLAY n filled {}

HOW TO DISPLAY board:
    IF board = {}:
        WRITE "No solution" /
    FOR p IN board:
        WRITE "# "^^(p-1), "O "
        WRITE "# "^^(#board - p) /
```

ABC: colons – based on user testing

```
HOW TO QUEENS n:
    DISPLAY n filled {}

HOW TO DISPLAY board:
    IF board = {}:
        WRITE "No solution" /
    FOR p IN board:
        WRITE "# "^^(p-1), "O "
        WRITE "# "^^(#board - p) /
```

Python: evolution

```
def lookup(caps, MIMEtype, key=None):
    entries = []
    if MIMEtype in caps:
        entries = entries + caps[MIMEtype]
    MIMEtypes = MIMEtype.split('/')
    MIMEtype = MIMEtypes[0] + '/*'
    if MIMEtype in caps:
        entries = entries + caps[MIMEtype]
    if key is not None:
        entries = [e for e in entries if key in e]
    return entries

def findparam(name, plist):
    name = name.lower() + '='
    n = len(name)
    for p in plist:
        if p[:n].lower() == name:
            return p[n:]
    return ''
```

People get worked up about syntax...

```
From:      (name redacted!)
To:        python-ideas@public.gmane.org
Subject:   Python Reviewed
Date:      Mon, 9 Jan 2017 19:25:45 +0800

...

The Bad:
     Colons at the end of if/while/for blocks. Most of the arguments in
favour of this decision boil down to PEP 20.2 "Explicit is better than
implicit". Well, no. if/while/for blocks are already explicit. Adding
the colon makes it doubly explicit and therefore redundant. There is no
reason I can see why this colon can't be made optional except for
possibly PEP20.13 "There should be one-- and preferably only one
--obvious way to do it". I don't agree that point is sufficient to
require colons.

...
```

Four days in and still no patch for critical "goto fail" bug in OS X (updated)

Critical crypto flaw takes on urgency as exploit code proliferates in the wild.

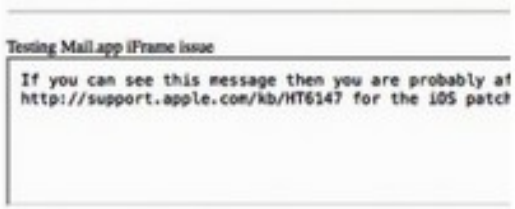
by Dan Goodin - Feb 25 2014, 5:45pm GMT

Update: Shortly after this brief went live, Apple released OS X version 10.9.2, which finally patches the critical "goto fail" bug.

It has been four days since Mac users began learning of a critical vulnerability in the latest version of OS X that gives attackers an easy way to surreptitiously circumvent the most widely used technology for preventing Internet eavesdropping. Three days ago, Apple told Reuters that it plans to release a patch "very soon," but it didn't elaborate on the details.

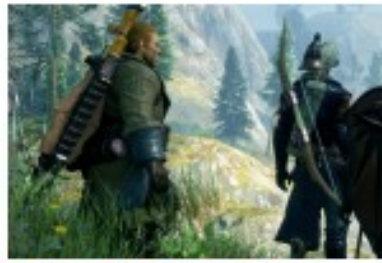
If it wasn't clear before, it should be painfully obvious now. The security and privacy of millions of Mavericks users

FURTHER READING



EXTREMELY CRITICAL CRYPTO FLAW IN IOS MAY ALSO AFFECT FULLY PATCHED MACS

LATEST FEATURE STORY



FEATURE STORY (2 PAGES)

Dragon Age: I
—Let's spend
together

Review: BioWare's mas
held together by its cor

WATCH ARS VIDEO

What went wrong?

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
                                 SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

Haskell: the offside rule

```
matMult x y     =  accumArray (+) 0 resultBounds
                              [((i,j), x!(i,k) * y!(k,j))
                                       | i <- range (li,ui),
                                         j <- range (lj',uj')
                                         k <- range (lj,uj)  ]

        where ((li,lj),(ui,uj))          = bounds x

              ((li',lj'),(ui',uj'))      = bounds y

              resultBounds
                | (lj,uj)==(li',ui')     = ((li,lj'),(ui,uj'))
                | otherwise              = error "incompatible”
```

Haskell: the offside rule

```
matMult x y     =  accumArray (+) 0 resultBounds
                              [((i,j), x!(i,k) * y!(k,j))
                                       | i <- range (li,ui),
                                         j <- range (lj',uj')
                                         k <- range (lj,uj)  ]

        where ((li,lj),(ui,uj))         = bounds x

              ((li',lj'),(ui',uj'))     = bounds y

              resultBounds
                | (lj,uj)==(li',ui')    = ((li,lj'),(ui,uj'))
                | otherwise             = error "incompatible”
```

BCPL: birth of...

```
let parse.command() be
$(  let token=?
    switchon scanner() into          //See what was typed.
    $(  case S.MOTION ... S.LEVEL:
            verb_scan.info    //Remember verb.
            token_nvscanner() //Next token.
            if token=S.CONT return    //Only said verb.
            if token=S.WITH\/token=S.AT token_nvscanner()
                        //ignore superfluous WITHs
            if token=S.MOTION nvscanner()
                        //Frig participles to verbs!
            try.it(@objct,token)  //Remember object.
            unless token=S.WITH\/token=S.AT token_nvscanner()
                        //Anything more?
            if token=S.AT at_true
            unless token=S.WITH\/token=S.AT\/token=S.CONT
            $(  let what=scan.info
                token_nvscanner()
                test S.IT le token le S.THEM then try.it(@objct,token) or
                test token=S.CONT\/token=S.WITH\/token=S.AT then
                  scan.info_what<>try.it(@objct,S.OBJECT)
                or try.it(@instrmnt,token)<>endcase
            $)
        if token=S.CONT instrmnt_0<>return  //No.
            if token=S.AT at_true
            token_nvscanner()
            try.it(@instrmnt,token)
            endcase
```

awk: significant ends of lines

```
BEGIN {
    FS = "\t"    # default
    OFS = FS
    while ((c = getopt(ARGC, ARGV, "sf:c:d:")) != -1) {
        if (c == "f") {
            by_fields = 1
            fieldlist = Optarg
        } else if (c == "c") {
            by_chars = 1
            fieldlist = Optarg
            OFS = ""
        } else if (c == "d") {
            if (length(Optarg) > 1) {
                printf("Using first character of %s" \
                " for delimiter\n", Optarg) > "/dev/stderr"
                Optarg = substr(Optarg, 1, 1)
            }
            FS = Optarg
            OFS = FS
            if (FS == " ")    # defeat awk semantics
                FS = "[ ]"
 ...
```

Preprocessing: C and Rust

```
#define SIOCGIFNAME  0x8910      /* get iface name    */
#define SIOCSIFLINK  0x8911      /* set iface channel    */
#define SIOCGIFCONF  0x8912      /* get iface list    */

#define fpclassify(x) \
     (sizeof (x) == sizeof (float) ? __fpclassifyf (x) : \
                                     __fpclassify (x))
```

```
macro_rules! foo {
    () => (fn x() { });
}

fn main() {
    foo!();
    x();
}
```

Any questions?

- Please be skeptical the next time you see a new programming language using C's syntax!
  - … have we really not learned anything since 1970?
  - … but: there are also advantages in familiarity...

- **Next week**: it's week 7 – no lectures or lab for CMP409; we will be back to normal in week 8