The Programming Language Landscape

Stephen A. Edwards

Columbia University

Fall 2014
The Diversity of Programming Languages

http://www.99-bottles-of-beer.net has programs in over 1,500 different programming languages and variations to generate the lyrics to the song “99 Bottles of Beer.”
99 Bottles of Beer

99 bottles of beer on the wall, 99 bottles of beer.
Take one down and pass it around, 98 bottles of beer on the wall.

98 bottles of beer on the wall, 98 bottles of beer.
Take one down and pass it around, 97 bottles of beer on the wall.

2 bottles of beer on the wall, 2 bottles of beer.
Take one down and pass it around, 1 bottle of beer on the wall.

1 bottle of beer on the wall, 1 bottle of beer.
Take one down and pass it around, no more bottles of beer on the wall.

No more bottles of beer on the wall, no more bottles of beer.
Go to the store and buy some more, 99 bottles of beer on the wall.
```java
class Bottles {
    public static void main(String args[]) {
        String s = "s";
        for (int beers=99; beers>-1;) {
            System.out.print(beers+" bottle"+s+" of beer on the wall, ");
            System.out.println(beers + " bottle" + s + " of beer, ");
            if (beers==0) {
                System.out.print("Go to the store, buy some more, ");
                System.out.println("99 bottles of beer on the wall.\n");
                System.exit(0);
            }
            else
                System.out.print("Take one down, pass it around, ");
                s = (--beers == 1)?"":s;
                System.out.println(beers+" bottle"+s+" of beer on the wall.\n");
        }
    }
}
```

Sean Russell,
class Bottles {
    public static void main(String[] args) {
        String s = "s";
        for (int beers = 99; beers > -1;)
            System.out.print(beers + " bottle" + s + " of beer on the wall, ");
        System.out.println(beers + " bottle" + s + " of beer, ");
        if (beers == 0) {
            System.out.print("Go to the store, buy some more, ");
            System.out.println("99 bottles of beer on the wall.");
            System.exit(0);
        } else
            System.out.print("Take one down, pass it around, ");
        s = (--beers == 1) ? "" : "s";
        System.out.println(beers + " bottle" + s + " of beer on the wall.");
    }
}

Gosling et al., Sun, 1991
Imperative, object-oriented, threaded
Based on C++, C, Algol, etc.
Statically typed
Automatic garbage collection
Architecturally neutral
Defined on a virtual machine (Java Bytecode)

Sean Russell,
```c
#define MAXBEER 99
void chug(int beers);

int main()
{
    int beers;
    for(beers = MAXBEER; beers; chug(beers--)) ;
    puts("\nTime to buy more beer!\n");
    return 0;
}

void chug(int beers)
{
    char howmany[8], *s;
    s = beers != 1 ? "s" : "";
    printf("%d bottle%s of beer on the wall,\n", beers, s);
    printf("%d bottle%s of beeeeer . . . ,\n", beers, s);
    printf("Take one down, pass it around,\n");
    if (--beers) sprintf(howmany, "%d", beers);
    else strcpy(howmany, "No more");
    s = beers != 1 ? "s" : "";
    printf("%s bottle%s of beer on the wall.\n", howmany, s);
}
```

#define MAXBEER 99

void chug(int beers);

int main()
{
    int beers;
    for(beers = MAXBEER; beers;
        puts("\nTime to buy more beer!
        return 0;
    }

void chug(int beers)
{
    char howmany[8], *s;
    s = beers != 1 ? "s" : "";
    printf("%d bottle%s of beer on the wall,
    printf("%d bottle%s of beer . . . ,
    printf("Take one down, pass it around,
    if (--beers) sprintf(howmany, "%d",
    else strcpy(howmany, "No more"
    s = beers != 1 ? "s" : "";
    printf("%s bottle%s of beer on the wall.
}
program ninetyninebottles
  integer bottles
  bottles = 99
1    format (I2, A)
2    format (A)
3    format (I2, A, /)
4    format (A, /)
10   write (*,1) bottles, ' bottles of beer on the wall,'
    write (*,1) bottles, ' bottles of beer.'
    write (*,2) 'Take one down, pass it around...'
    if (bottles - 1 .gt. 1) then
        write (*,3) bottles - 1, ' bottles of beer on the wall.'
    else
        write (*,3) bottles - 1, ' bottle of beer on the wall.'
    end if
    bottles = bottles - 1
7    if (bottles - 1) 30, 20, 10
*    Last verse
20   write (*,1) bottles, ' bottle of beer on the wall,'
    write (*,1) bottles, ' bottle of beer.'
    write (*,2) 'Take one down, pass it around...'
    write (*,2) 'No bottles of beer on the wall.'
30   stop
end

program ninetyninebottles
integer bottles
bottles = 99
1 format (I2, A)
2 format (A)
3 format (I2, A, /)
4 format (A, /)
10 write (*,1) bottles, ' bottles of beer on the wall,'
   write (*,1) bottles, ' bottles of beer.'
   write (*,2) 'Take one down, pass it around...'
   if (bottles - 1 .gt. 1)
      write (*,3) bottles - 1, ' bottles of beer on the wall.'
   else
      write (*,3) bottles - 1, ' bottle of beer on the wall.'
   end if
   bottles = bottles - 1
   if (bottles - 1) 30, 20, 10
  * Last verse
20 write (*,1) bottles, ' bottle of beer on the wall,'
   write (*,1) bottles, ' bottle of beer.'
   write (*,2) 'Take one down, pass it around...'
   write (*,4) 'No bottles of beer on the wall.'
30 stop
end

Backus, IBM, 1956
Imperative language for science and engineering
First compiled language
Fixed format lines (for punch cards)
Arithmetic expressions, If, Do, and Goto statements
Scalar (number) and array types
Limited string support
Still common in high-performance computing
Inspired most modern languages, especially BASIC
BEGIN {
    for (i = 99; i >= 0; i--) {
        print ubottle(i), "on the wall," , lbottle(i) "." 
        print action(i), lbottle(inext(i)), "on the wall."
        print
    }
}

function ubottle(n) {
    return sprintf("%s bottle%s of beer", n?n:"No more", n-1?s":"")
}

function lbottle(n) {
    return sprintf("%s bottle%s of beer", n?n:"no more", n-1?s":"")
}

function action(n) {
    return sprintf("%s", n ? "Take one down and pass it around," : "Go to the store and buy some more," )
}

function inext(n) {
    return n ? n - 1 : 99
}
BEGIN {
    for (i = 99; i >= 0; i--) {
        print ubottle(i), "on the wall,", lbottle(i) ".";
        print action(i), lbottle(inext(i)), "on the wall."
    }
}

function ubottle(n) {
    return sprintf("%s bottle%s of beer", n ? n : "No more", n - 1 ? "s": "")
}

function lbottle(n) {
    return sprintf("%s bottle%s of beer", n ? n : "no more", n - 1 ? "s": "")
}

function action(n) {
    return sprintf("%s", n ? "Take one down and pass it around,": "Go to the store and buy some more," )
}

function inext(n) {
    return n ? n - 1 : 99
}

Aho, Weinberger, and Kernighan, Bell Labs, 1977

Interpreted domain-specific scripting language for text processing

Pattern-action statements matched against input lines

C-inspired syntax

Automatic garbage collection

OsamuAoki,
AWK (bottled version)

Wilhelm Weske,
for quant in range(99, 0, -1):
    if quant > 1:
        print quant, "bottles of beer on the wall,", \
        quant, "bottles of beer."
    if quant > 2:
        suffix = str(quant - 1) + " bottles of beer on the wall."
    else:
        suffix = "1 bottle of beer on the wall."
    elif quant == 1:
        print "1 bottle of beer on the wall, 1 bottle of beer."
        suffix = "no more beer on the wall!"
    print "Take one down, pass it around," , suffix
    print ""

Gerold Penz,
for quant in range(99, 0, -1):
    if quant > 1:
        print quant, "bottles of beer on the wall,",
        print quant, "bottles of beer."
    if quant > 2:
        suffix = str(quant - 1) + " bottles of beer on the wall."
    else:
        suffix = "1 bottle of beer on the wall."
    elif quant == 1:
        print "1 bottle of beer on the wall,"
        print "1 bottle of beer."
        suffix = "no more beer on the wall!"
    print "Take one down, pass it around,"
    print ""

Guido van Rossum, 1989
Object-oriented, imperative
General-purpose scripting language
Indentation indicates grouping
Dynamically typed
Automatic garbage collection

Gerold Penz,
APL (A Programming Language)

Program written by JT. Taylor, www.jttaylor.net

T1←98↑[1]∅1 99πl99

T4←∅1 98πl98

T1,(98 30π’ BOTTLES OF BEER ON THE WALL, ‘),T1, (98 47π’BOTTLES OF BEER, TAKE ONE DOWN, PASS IT AROUND,’),T4,(98 28π’BOTTLES OF BEER ON THE WALL ,’)

‘1 BOTTLE OF BEER ON THE WALL, 1 BOTTLE OF BEER, TAKE IT DOWN, PASS IT AROUND, NO BOTTLES OF BEER ON THE WALL.’

APL

- APL (A Programming Language)
- Program written by J.T. Williams

```
T1←98^[1]∅1 99π99
T4←∅1 98π98
T1,(98 30π’BOTTLES OF BEER,
(98 47π’BOTTLES OF BEER,
AROUND,’),T4,(98 28π’BOTTLE
WALL,’)

‘1 BOTTLE OF BEER ON THE WALL,
TAKE IT DOWN, PASS IT AROUND,
ON THE WALL.’
```

Iverson, IBM, 1960
Imperative, matrix-centric
E.g., perform an operation on each element of a vector
Uses own specialized character set
Concise, effectively cryptic
Primarily symbols instead of words
Dynamically typed
Odd left-to-right evaluation policy
Useful for statistics, other matrix-oriented applications

: bottles ( n -- n-1 )
  dup 1 = IF  "." One bottle of beer on the wall," CR
  "." One bottle of beer," CR
  "." Take it down,"
  ELSE  dup . "." bottles of beer on the wall," CR
  dup . "." bottles of beer," CR
  "." Take one down,"
  THEN
  CR
  "." Pass it around," CR
  1-
  ?dup IF  dup 1 = IF  "." One bottle of beer on the wall;" 
  ELSE  dup . "." bottles of beer on the wall;" 
  THEN
  ELSE  . "." No more bottles of beer on the wall." 
  THEN
  CR
;

: nbottles ( n -- )
BEGIN  bottles  ?dup NOT UNTIL ;

99 nbottles

Dan Reish,
FORTH

: \texttt{bottles} ( \texttt{n \-- n-1} )
  \texttt{dup} \texttt{1 = IF } \texttt{." One bottle of beer on the wall,"}
  \texttt{." One bottle of beer,"}
  \texttt{." Take it down,"}
  \texttt{ELSE \texttt{dup} \texttt{.} \texttt{" bottles of beer on the wall,"}
  \texttt{\texttt{dup} \texttt{.} \texttt{" bottles of beer,"}
  \texttt{.} \texttt{" Take one down,"}
  \texttt{\texttt{THEN}}
  \texttt{\texttt{CR}}
  \texttt{\texttt{.} \texttt{" Pass it around,"} \texttt{\texttt{CR}}
  \texttt{1-} \texttt{\?dup} \texttt{\texttt{IF} \texttt{\texttt{dup} \texttt{1 = IF} \texttt{.}" One bottle of beer on the wall;}
  \texttt{ELSE \texttt{dup} \texttt{.} \texttt{" bottles of beer on the wall;}
  \texttt{\texttt{THEN}}
  \texttt{\texttt{ELSE} \texttt{.} \texttt{" No more bottles of beer on the wall."}
  \texttt{\texttt{THEN}}
  \texttt{\texttt{CR}}
  \texttt{;}

: \texttt{nbottles} ( \texttt{n -- } )
  \texttt{BEGIN} \texttt{.\texttt{bottles} \ ?\texttt{dup} NOT \texttt{UNTIL} \texttt{;}

99 \texttt{nbottles}

Moore, NRAO, 1973

Stack-based imperative language
Trivial, RPN-inspired grammar
Easily becomes cryptic
Untyped
Low-level, very lightweight
Highly extensible: easy to make programs compile themselves
Used in some firmware boot systems (Apple, IBM, Sun)
Inspired the PostScript language for laser printers

Dan Reish,
Imperative, stack-based language

Space, Tab, and Line Feed characters only

Number literals in binary: Space=0, Tab=1, LF=end

Less-than-programmer-friendly syntax; reduces toner consumption
bottles :-
    bottles(99).

bottles(1) :-
    write('1 bottle of beer on the wall, 1 bottle of beer,'), nl,
    write('Take one down, and pass it around,'), nl,
    write('Now they are all gone.'), nl, !.

bottles(X) :-
    write(X), write(' bottles of beer on the wall,'), nl,
    write(X), write(' bottles of beer,'), nl,
    write('Take one down and pass it around,'), nl,
    NX is X - 1,
    write(NX), write(' bottles of beer on the wall.'), nl, nl,
    bottles(NX).

Remko Trocon et al.,
Prolog

Alain Colmerauer et al., 1972
Logic programming language
Programs are relations: facts and rules
Program execution consists of trying to satisfy queries
Designed for natural language processing, expert systems, and theorem proving

Remko Trocon et al.,
SELECT  
CASE (bottlecount)  
  WHEN 0 THEN 'No more bottle of beer on the wall, no more bottles of beer on the wall. '  
  'Go to the store and buy some more, 99 bottles of beer on the wall.'  
  WHEN 1 THEN '1 bottle of beer on the wall, 1 bottle of beer. ' ||  
  'Take one down and pass it around, no more bottles of beer on the wall.'  
  WHEN 2 THEN '2 bottles of beer on the wall, 2 bottles of beer. ' ||  
  'Take one down and pass it around, 1 bottle of beer on the wall.'  
  ELSE  
  rtrim (cast((BottleCount) as char(2))) || ' bottles of beer on the wall, '  
  rtrim (cast((BottleCount) as char(2))) || ' bottles of beer. '  
  rtrim (cast((BottleCount)-1 as char(2))) || ' bottles of beer on the wall.'  
END  
FROM  
(  
  SELECT avalue * 10 + bvalue as bottlecount  
  FROM  
  (VALUES (9), (8), (7), (6), (5), (4), (3), (2), (1), (0)) a(avalue)  
  (VALUES (9), (8), (7), (6), (5), (4), (3), (2), (1), (0)) b(bvalue)  
) as valuelist;
SELECT  
CASE (bottlecount)
  WHEN 0 THEN 'No more bottle of beer on the wall, no more bottles of beer.
  Go to the store and buy some more, 99 bottles of beer on the wall.' ||
  WHEN 1 THEN '1 bottle of beer on the wall, 1 bottle of beer.
  Take one down and pass it around, no more bottles of beer on the wall.' ||
  WHEN 2 THEN '2 bottles of beer on the wall, 2 bottles of beer.
  Take one down and pass it around, 1 bottle of beer on the wall.' ||
  ELSE rtrim (cast((BottleCount) as char(2))) || ' bottles of beer on the wall, ' ||
      rtrim (cast((BottleCount) as char(2))) || ' bottles of beer. ' ||
      rtrim (cast((BottleCount)-1 as char(2))) || ' bottles of beer on the wall.'
END
FROM
  (SELECT avalue * 10 + bvalue AS bottlecount
   FROM
     (VALUES (9), (8), (7), (6), (5), (4), (3), (2), (1), (0)) as a
     , (VALUES (9), (8), (7), (6), (5), (4), (3), (2), (1), (0)) as b)
   AS valuelist;

Chamberlin and Boyce, IBM, 1974
Declarative language for databases
Semantics based on the relational model
Queries on tables: select with predicates, joining, aggregating
Database query optimization: declaration to procedure

Kent Olsen,
(defun bottles-of-bier (n)
  (case n
    (0
     '(
       "No more bottles of beer on the wall no more bottles of beer."
       "Go to the store and buy some more 99 bottles of beer on the wall."
     ))
    (1
     '(
       "1 bottle of beer on the wall 1 bottle of beer."
       "Take one down and pass it around no more bottles of beer on the wall."
     ,@(bottles-of-bier 0))))
    (2
     '(
       "2 bottles of beer on the wall 2 bottles of beer."
       "Take one down and pass it around 1 bottle of beer on the wall."
     ,@(bottles-of-bier 1))))
    (t
     '(
       "n bottles of beer on the wall ,n bottles of beer."
       "Take one down and pass it around"
       ,(1- n) "bottles of beer on the wall."
     ,@(bottles-of-bier (1- n))))))

(defun bottles-of-beer (n)
  (case n
    (0 '/gif)
    (1 '('1 bottle of beer on the wall
      'Take one down and pass it around
      'no more bottles of beer on the wall."
      ,(bottles-of-beer 0)))
    (2 '('2 bottles of beer on the wall
      'Take one down and pass it around
      '1 bottle of beer on the wall."
      ,(bottles-of-beer 1)))
    (t '('n bottles of beer on the wall
      'Take one down and pass it around
      ,(1- n) bottles of beer on the wall"
      ,(bottles-of-beer (1- n))))
  ))

McCarthy, MIT, 1958

Functional: recursive, list-focused functions

Semantics from Church’s Lambda Calculus

Simple, heavily parenthesized S-expression syntax

Dynamically typed

Automatic garbage collection

Originally for AI applications

Dialects: Scheme and Common Lisp

Haskell

```haskell
bottles :: Int -> String
bottles n
    | n == 0 = "no more bottles"
    | n == 1 = "1 bottle"
    | n > 1 = show n ++ " bottles"

verse :: Int -> String
verse n
    | n == 0 = "No more bottles of beer on the wall, "
      ++ "no more bottles of beer.\n"
      ++ "Go to the store and buy some more, "
      ++ "99 bottles of beer on the wall."
    | n > 0 = bottles n ++ " of beer on the wall, "
      ++ bottles n
      ++ " of beer.\n"
      ++ "Take one down and pass it around, "
      ++ bottles (n-1) ++ " of beer on the wall.\n"

main = mapM (putStrLn . verse) [99,98..0]
```

Simon Johansson,
Haskell

```haskell
bottles :: Int -> String
bottles n
  | n == 0 = "no more bottles"
  | n == 1 = "1 bottle"
  | n > 1 = show n ++ " bottles"

verse :: Int -> String
verse n
  | n == 0 = "No more bottles of beer on the wall, 
                     "no more bottles of beer.
        "Go to the store and buy some more, 
        "99 bottles of beer on the wall."
  | n > 0 = bottles n ++ " of beer on the wall, 
        ++ bottles n ++ " of beer.
        "Take one down and pass it around, 
        ++ bottles (n-1) ++ " of beer on the wall.""

main = mapM (putStrLn . verse) [99,98..0]
```

Peyton Jones et al., 1990

Functional

Pure: no side-effects

Lazy: computation only on demand; infinite data structures

Statically typed; types inferred

Algebraic data types, pattern matching, lists, strings

Great for compilers, domain-specific languages, type system research

Related to ML, OCaml