CHAPTER 2

Evolution of the Major Programming Languages
Genealogy of Common Languages
Zuse’s Plankalkül

- Designed in 1945, but not published until 1972
- Never implemented
- Advanced data structures
  - floating point, arrays, records
- Invariants
Plankalkül Syntax


<table>
<thead>
<tr>
<th></th>
<th>A + 1 =&gt; A</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>S</td>
<td>1.n</td>
</tr>
<tr>
<td></td>
<td>1.n</td>
</tr>
</tbody>
</table>

(subscripts) (data types)
Minimal Hardware Programming: Pseudocodes

• What was wrong with using machine code?
  • Poor readability
  • Poor modifiability
  • Expression coding was tedious
  • Machine deficiencies--no indexing or floating point
Pseudocodes: Short Code

• Short Code developed by Mauchly in 1949 for BINAC computers
  • Expressions were coded, left to right
  • Example of operations:

01 -  06 abs value  1n (n+2)nd power
02 )  07 +          2n (n+2)nd root
03 =  08 pause       4n if <= n
04 /  09 (           58 print and tab
Pseudocodes: Speedcoding

• Speedcoding developed by Backus in 1954 for IBM 701
  • Pseudo ops for arithmetic and math functions
  • Conditional and unconditional branching
  • Auto-increment registers for array access
  • Slow!
  • Only 700 words left for user program
Pseudocodes: Related Systems

• The UNIVAC Compiling System
  • Developed by a team led by Grace Hopper
  • Pseudocode expanded into machine code

• David J. Wheeler (Cambridge University)
  • developed a method of using blocks of re-locatable addresses to solve the problem of absolute addressing
IBM 704 and Fortran

- Fortran 0: 1954 - not implemented
- Fortran I: 1957
  - Designed for the new IBM 704, which had index registers and floating point hardware
    - This led to the idea of compiled programming languages, because there was no place to hide the cost of interpretation (no floating-point software)

- Environment of development
  - Computers were small and unreliable
  - Applications were scientific
  - No programming methodology or tools
  - Machine efficiency was the most important concern
Design Process of Fortran

• Impact of environment on design of Fortran I
  • No need for dynamic storage
  • Need good array handling and counting loops
  • No string handling, decimal arithmetic, or powerful input/output (for business software)
Fortran I Overview

• First implemented version of Fortran
  • Names could have up to six characters
  • Post-test counting loop (DO)
  • Formatted I/O
  • User-defined subprograms
  • Three-way selection statement (arithmetic IF)
  • No data typing statements
Fortran I Overview (continued)

• First implemented version of FORTRAN
  • No separate compilation
  • Compiler released in April 1957, after 18 worker-years of effort
  • Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of 704
  • Code was very fast
  • Quickly became widely used
Fortran II

• Distributed in 1958
  • Independent compilation
  • Fixed the bugs
Fortran IV

- Evolved during 1960-62
  - Explicit type declarations
  - Logical selection statement
  - Subprogram names could be parameters
  - ANSI standard in 1966
Fortran 77

• Became the new standard in 1978
  • Character string handling
  • Logical loop control statement
  • **IF–THEN–ELSE** statement
Fortran 90

• Most significant changes from Fortran 77
  • Modules
  • Dynamic arrays
  • Pointers
  • Recursion
  • \texttt{CASE} statement
  • Parameter type checking
module module1
integer:: n
contains

recursive subroutine sub1(x)
integer,intent(inout):: x
integer:: y
y = 0
if (x < n) then
  x = x + 1
  y = x**2
  print *, 'x = ', x, ', y = ', y
  call sub1(x)
  print *, 'x = ', x, ', y = ', y
end if
end subroutine sub1

end module module1

program main
use module1
integer:: x = 0
print *, 'Enter number of repeats'
read (*,*) n
call sub1(x)
end program main
Latest versions of Fortran

• Fortran 95 – relatively minor additions, plus some deletions
• Fortran 2003 – support for OOP, procedure pointers, interoperability with C
• Fortran 2008 – blocks for local scopes, co-arrays, Do Concurrent
Fortran Evaluation

• Highly optimizing compilers (all versions before 90)
  • Types and storage of all variables are fixed before run time
• Dramatically changed forever the way computers are used
Functional Programming: LISP

- LIST Processing language
  - Designed at MIT by McCarthy
- AI research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)
- Only two data types: atoms and lists
- Syntax is based on lambda calculus
LISP Example

(print "Hello world")

(defun factorial (n)
  (if (<= n 1)
      1
      (* n (factorial (- n 1)))))
Representation of Two LISP Lists

Representing the lists \((A \ B \ C \ D)\) and \((A \ (B \ C) \ D \ (E \ (F \ G)))\)
LISP Evaluation

- Pioneered functional programming
  - No need for variables or assignment
  - Control via recursion and conditional expressions
- Still the dominant language for AI
- COMMON LISP and Scheme are contemporary dialects of LISP
- ML, Haskell, and F# are also functional programming languages, but use very different syntax
Haskell Example

double x = x+x
quadruple x = double(double x)
factorial n=product[1..n]
average ns=div(sum ns)(length ns)

qsort[]=[]
qsort(x:xs)=qsort smaller++[x]++qsort larger
    where
        smaller=[a|a <- xs, a<=x]
        larger=[b|b <- xs, b>x]

splitAt :: Int->[a]->([a],[a])
splitAt n xs=(take n xs, drop n xs)
Scheme

• Developed at MIT in mid 1970s
• Small
• Extensive use of static scoping
• Functions as first-class entities
• Simple syntax (and small size) make it ideal for educational applications
COMMON LISP

• An effort to combine features of several dialects of LISP into a single language
• Large, complex, used in industry for some large applications
The First Step Toward Sophistication: ALGOL 60

• Environment of development
  • FORTRAN had (barely) arrived for IBM 70x
  • Many other languages were being developed, all for specific machines
  • No portable language; all were machine-dependent
  • No universal language for communicating algorithms

• ALGOL 60 was the result of efforts to design a universal language
Early Design Process

• ACM and GAMM met for four days for design (May 27 to June 1, 1958)

• Goals of the language
  • Close to mathematical notation
  • Good for describing algorithms
  • Must be translatable to machine code
ALGOL 58

• Concept of type was formalized
• Names could be any length
• Arrays could have any number of subscripts
• Parameters were separated by mode (in & out)
• Subscripts were placed in brackets
• Compound statements (\texttt{begin \ldots end})
• Semicolon as a statement separator
• Assignment operator was :=
• \texttt{if} had an \texttt{else-if} clause
• No I/O – “would make it machine dependent”
ALGOL 58 Implementation

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)
- Although IBM was initially enthusiastic, all support was dropped by mid 1959
ALGOL 60 Overview

- Modified ALGOL 58 at 6-day meeting in Paris
- New features
  - Block structure (local scope)
  - Two parameter passing methods
  - Subprogram recursion
  - Stack-dynamic arrays
- Still no I/O and no string handling
procedure Absmax(a) Size:=(n, m) Result:=(y) Subscripts:=(i, k);
    value n, m; array a; integer n, m, i, k; real y;
comment The absolute greatest element of the matrix a, of size n by m
is transferred to y, and the subscripts of this element to i and k;
begin integer p, q;
    y := 0; i := k := 1;
    for p:=1 step 1 until n do
        for q:=1 step 1 until m do
            if abs(a[p, q]) > y then
                begin y := abs(a[p, q]);
                    i := p; k := q
                end
end Absmax
ALGOL 60 Evaluation

• **Successes**
  • It was the standard way to publish algorithms for over 20 years
  • All subsequent imperative languages are based on it
  • First machine-independent language
  • First language whose syntax was formally defined (BNF)
ALGOL 60 Evaluation (continued)

• Failure
  • Never widely used, especially in U.S.
  • Reasons
    • Lack of I/O and the character set made programs non-portable
    • Too flexible--hard to implement
    • Entrenchment of Fortran
    • Formal syntax description
    • Lack of support from IBM
Computerizing Business Records: COBOL

• Environment of development
  • UNIVAC was beginning to use FLOW-MATIC
  • USAF was beginning to use AIMACO
  • IBM was developing COMTRAN
COBOL Historical Background

• Based on FLOW-MATIC
• FLOW-MATIC features
  • Names up to 12 characters, with embedded hyphens
  • English names for arithmetic operators (no arithmetic expressions)
  • Data and code were completely separate
  • The first word in every statement was a verb
COBOL Design Process

• First Design Meeting (Pentagon) - May 1959

• Design goals
  • Must look like simple English
  • Must be easy to use, even if that means it will be less powerful
  • Must broaden the base of computer users
  • Must not be biased by current compiler problems

• Design committee members were all from computer manufacturers and DoD branches

• Design Problems: arithmetic expressions? subscripts? Fights among manufacturers
$ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM-ID. Multiplier.
AUTHOR. Michael Coughlan.
* Example program using ACCEPT, DISPLAY and MULTIPLY to
* get two single digit numbers from the user and multiply them together

DATA DIVISION.

WORKING-STORAGE SECTION.

01 Num1 PIC 9 VALUE ZEROS.
01 Num2 PIC 9 VALUE ZEROS.
01 Result PIC 99 VALUE ZEROS.

PROCEDURE DIVISION.
   DISPLAY "Enter first number (1 digit) : " WITH NO ADVANCING.
   ACCEPT Num1.
   DISPLAY "Enter second number (1 digit) : " WITH NO ADVANCING.
   ACCEPT Num2.
   MULTIPLY Num1 BY Num2 GIVING Result.
   DISPLAY "Result is = ", Result.
   STOP RUN.
COBOL Evaluation

• Contributions
  • First macro facility in a high-level language
  • Hierarchical data structures (records)
  • Nested selection statements
  • Long names (up to 30 characters), with hyphens
  • Separate data division
COBOL: DoD Influence

- First language required by DoD
  - would have failed without DoD
- Still the most widely used business applications language
The Beginning of Timesharing: BASIC

• Designed by Kemeny & Kurtz at Dartmouth
• Design Goals:
  • Easy to learn and use for non-science students
  • Must be “pleasant and friendly”
  • Fast turnaround for homework
  • Free and private access
  • User time is more important than computer time
• Current popular dialect: Visual BASIC
• First widely used language with time sharing
BASIC Example

```
INPUT "What is your name: ", UserName$
PRINT "Hello "; UserName$
DO
   INPUT "How many stars do you want: ", NumStars
   Stars$ = STRING$(NumStars, "*")
   PRINT Stars$
   DO
      INPUT "Do you want more stars? ", Answer$
      LOOP UNTIL Answer$ <> ""
      Answer$ = LEFT$(Answer$, 1)
      LOOP WHILE UCASE$(Answer$) = "Y"
   PRINT "Goodbye "; UserName$
```

2.8 Everything for Everybody: PL/I

- Designed by IBM and SHARE
- Computing situation in 1964 (IBM's point of view)
  - Scientific computing
    - IBM 1620 and 7090 computers
    - FORTRAN
    - SHARE user group
  - Business computing
    - IBM 1401, 7080 computers
    - COBOL
    - GUIDE user group
PL/I: Background

• By 1963
  • Scientific users began to need more elaborate I/O, like COBOL had; business users began to need floating point and arrays for MIS
  • It looked like many shops would begin to need two kinds of computers, languages, and support staff--too costly

• The obvious solution
  • Build a new computer to do both kinds of applications
  • Design a new language to do both kinds of applications
PL/I: Design Process

• Designed in five months by the 3 X 3 Committee
  • Three members from IBM, three members from SHARE
• Initial concept
  • An extension of Fortran IV
• Initially called NPL (New Programming Language)
• Name changed to PL/I in 1965
/* Read in a line, which contains a string, 
/* and then print every subsequent line that contains that string. */

find_strings: procedure options (main);
   declare pattern character (100) varying;
   declare line character (100) varying;
   declare (line_no, end_file) fixed binary;

   end_file = 0;
on endfile (sysin) end_file = 1;

   get edit (pattern) (L);
   line_no = 1;
do while (end_file = 0);
      if index(line, pattern) > 0 then
         put skip list (line_no, line);
         line_no = line_no + 1;
      get edit (line) (L);
   end;

end find_strings;
PL/I: Evaluation

• PL/I contributions
  • First unit-level concurrency
  • First exception handling
  • Switch-selectable recursion
  • First pointer data type
  • First array cross sections

• Concerns
  • Many new features were poorly designed
  • Too large and too complex
Two Early Dynamic Languages: APL and SNOBOL

• Characterized by dynamic typing and dynamic storage allocation
• Variables are untyped
  • A variable acquires a type when it is assigned a value
• Storage is allocated to a variable when it is assigned a value
APL Example

[6] L←(Lσ' ':' )↓L←,L
   ∘ drop To:
[7] L←LJUST VTOM',',L
   ∘ mat with one entry per row
[8] S←1++/\\L≠'( '
   ∘ length of address
[9] X←0↓/\S
[10] L←S↑(-(ρL)+0,X)↑L
    ∘ align the (names)
    ∘ address
[12] N←0 1↓DLTB(0,X)↑L
    ∘ names)
[13] N←,' α',N
[14] N[(N=' _')/σρN]←'
    ∘ change _ to blank
[15] N←0 1↓RJUST VTOM N
    ∘ names
[16] S←+/\\', '≠σN
    ∘ length of last word in name
APL: A Programming Language

- Designed as a hardware description language at IBM by Ken Iverson around 1960
  - Highly expressive (many operators, for both scalars and arrays of various dimensions)
  - Programs are very difficult to read (throw away programming)
- Still in use; minimal changes
SNOBOL

• Designed as a string manipulation language at Bell Labs by Farber, Griswold, and Polensky in 1964
• Powerful operators for string pattern matching
• Slower than alternative languages (and thus no longer used for writing editors)
• Still used for certain text processing tasks
ASK
  OUTPUT = "Your name? 
  NAME = INPUT :F(DONE)
  OUTPUT = "Hello " NAME :(ASK)
DONE
  OUTPUT = "Finished"
END

LETTER = "abcdefgijklmnopqrstuvwxyz"
LETTERORDOT = "." LETTER
LETTERORSLASH = "/" LETTER

LINE = INPUT
LINE SPAN(LETTER) . PROTO ":/" SPAN(LETTERORDOT) . HOST "/" SPAN(LETTERORSLASH) . RES

OUTPUT = PROTO
OUTPUT = HOST
OUTPUT = RES
END
The Beginning of Data Abstraction: SIMULA 67

• Designed primarily for system simulation in Norway by Nygaard and Dahl
• Based on ALGOL 60 and SIMULA I
• Primary Contributions
  • Coroutines - a kind of subprogram
  • Classes, objects, and inheritance
SIMULA Example

Begin
Class Glyph;
    Virtual: Procedure print Is Procedure print;
Begin
End;

Glyph Class Char (c);
    Character c;
Begin
    Procedure print;
    OutChar(c);
End;

Glyph Class Line (elements);
    Ref (Glyph) Array elements;
Begin
    Procedure print;
    Begin
        Integer i;
        For i := 1 Step 1 Until UpperBound (elements, 1) Do
            elements (i).print;
            OutImage;
    End;
End;

Ref (Glyph) rg;
Ref (Glyph) Array rgs (1 : 4);

! Main program;
rgs (1):= New Char ('A');
rgs (2):= New Char ('b');
rgs (3):= New Char ('b');
rgs (4):= New Char ('a');
rg:=- New Line (rgs);
rg.print;
End;
Orthogonal Design: ALGOL 68

- From the continued development of ALGOL 60 but not a superset of that language
- Source of several new ideas (even though the language itself never achieved widespread use)
- Design is based on the concept of orthogonality
  - A few basic concepts, plus a few combining mechanisms
ALGOL 68 Orthogonality Examples

int i; real r; [1:1] int rowi; ref int refi;
union(int, real) ir; proc int p;

r:= i/r     -- i gets widened
ir:= i;     -- uniting
ir:= r;     -- uniting
i:= p;      -- deproceduring;
i:= refi;   -- dereferencing (twice)
p;          -- deproceduring; voiding
rowi:= 5;   -- rowing
ALGOL 68 Evaluation

• Contributions
  • User-defined data structures
  • Reference types
  • Dynamic arrays (called flex arrays)

• Comments
  • Less usage than ALGOL 60
  • Had strong influence on subsequent languages, especially Pascal, C, and Ada
Pascal - 1971

- Developed by Wirth (a former member of the ALGOL 68 committee)
- Designed for teaching structured programming
- Small, simple, nothing really new
- Largest impact was on teaching programming
  - From mid-1970s until the late 1990s, it was the most widely used language for teaching programming
Pascal Examples

type
  a = Array [1..10] of Integer;
b = record
  x: Integer;
y: Char
end;
c = File of a;

while a <> b do writeln('Waiting');
if a > b then
  writeln('Condition met')
else
  writeln('Condition not met');

for i := 1 to 10 do
  writeln('Iteration: ', i:1);

repeat
  a := a + 1
until a = 10;

case i of
  0: write('zero');
  1: write('one');
  2: write('two')
end;
C - 1972

- Designed for systems programming (at Bell Labs by Dennis Ritchie)
- Evolved primarily from BCLP and B, but also ALGOL 68
- Powerful set of operators, but poor type checking
- Initially spread through UNIX
- Though designed as a systems language, it has been used in many application areas
C Examples

```c
#include <stdio.h>

int main(void)
{
    int first, second;

    printf("Enter two integers > ");
    scanf("%d %d", &first, &second);
    printf("The two numbers are: %d %d\n", first, second);
    printf("Their sum is %d\n", first+second);
}

/* strncmp: compare at most n characters of t with s */
int strncmp(char *s, char *t, int n)
{
    for( ; *s == *t; s++, t++)
        if(*s == '\0' || --n <= 0)
            return 0;
    return *s - *t;
}
```
Programming Based on Logic: Prolog

• Developed, by Comerauer and Roussel (University of Aix-Marseille), with help from Kowalski (University of Edinburgh)
• Based on formal logic
• Non-procedural
• Can be summarized as being an intelligent database system that uses an inferencing process to infer the truth of given queries
• Comparatively inefficient
• Few application areas
Prolog Example

mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).
sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).
parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).

This results in the following query being evaluated as true:

?- sibling(sally, erica).
Yes
History’s Largest Design Effort: Ada

- Huge design effort, involving hundreds of people, much money, and about eight years
- Sequence of requirements (1975-1978)
  - (Strawman, Woodman, Tinman, Ironman, Steelman)
- Named Ada after Augusta Ada Byron, the first programmer
Ada Example

```ada
while a /= b loop
  Ada.Text_IO.Put_Line ("Waiting");
end loop;

if a > b then
  Ada.Text_IO.Put_Line ("Condition met");
else
  Ada.Text_IO.Put_Line ("Condition not met");
end if;

for i in 1 .. 10 loop
  Ada.Text_IO.Put ("Iteration: ");
  Ada.Text_IO.Put (i);
  Ada.Text_IO.Put_Line;
end loop;

loop
  a := a + 1;
  exit when a = 10;
end loop;

case i is
  when 0 => Ada.Text_IO.Put("zero");
  when 1 => Ada.Text_IO.Put("one");
  when 2 => Ada.Text_IO.Put("two");
  -- case statements have to cover all possible cases:
  when others => Ada.Text_IO.Put("none of the above");
end case;
```
Ada Evaluation

• Contributions
  • Packages - support for data abstraction
  • Exception handling - elaborate
  • Generic program units
  • Concurrency - through the tasking model

• Comments
  • Competitive design
  • Included all that was then known about software engineering and language design
  • First compilers were very difficult; the first really usable compiler came nearly five years after the language design was completed
Ada 95

- Ada 95 (began in 1988)
  - Support for OOP through type derivation
  - Better control mechanisms for shared data
  - New concurrency features
  - More flexible libraries

- Ada 2005
  - Interfaces and synchronizing interfaces

- Popularity suffered because the DoD no longer requires its use but also because of popularity of C++
Object-Oriented Programming: Smalltalk

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg
- First full implementation of an object-oriented language (data abstraction, inheritance, and dynamic binding)
- Pioneered the graphical user interface design
- Promoted OOP
Smalltalk Examples

```smalltalk
| rectangles aPoint collisions |
rectangles := OrderedCollection
    with: (Rectangle left: 0 right: 10 top: 100 bottom: 200)
    with: (Rectangle left: 10 right: 10 top: 110 bottom: 210).
aPoint := Point x: 20 y: 20.
collisions := rectangles select: [:aRect | aRect containsPoint: aPoint].
```

draw

```smalltalk
"draw the rectangle"
Transcript show:
 'Drawing a rectangle at:(' ,
 self x printString , '),' ,
 self y printString ,
 ' ), width ' ,
 self width printString ,
 ' , height ' ,
 self height printString;
cr.
```
Combining Imperative and Object-Oriented Programming: C++

• Developed at Bell Labs by Stroustrup in 1980
• Evolved from C and SIMULA 67
• Facilities for object-oriented programming, taken partially from SIMULA 67
• A large and complex language, in part because it supports both procedural and OO programming
• Rapidly grew in popularity, along with OOP
• ANSI standard approved in November 1997
• Microsoft’s version: MC++
  • Properties, delegates, interfaces, no multiple inheritance
C++ Examples

class calc
{
  public:
    int multiply(int x, int y);
    int add(int x, int y);
};
int calc::multiply(int x, int y)
{
  return x*y;
}
int calc::add(int x, int y)
{
  return x+y;
}

template <class A_Type> class calc
{
  public:
    A_Type multiply(A_Type x, A_Type y);
    A_Type add(A_Type x, A_Type y);
};
template <class A_Type> A_Type calc<A_Type>::multiply(A_Type x, A_Type y)
{
  return x*y;
}
template <class A_Type> A_Type calc<A_Type>::add(A_Type x, A_Type y)
{
  return x+y;
}

calc <double> a_calc_class;
Related OOP Languages

- **Objective-C** (designed by Brad Cox – early 1980s)
  - C plus support for OOP based on Smalltalk
  - Uses Smalltalk’s method calling syntax
  - Used by Apple for systems programs
- **Delphi** (Borland)
  - Pascal plus features to support OOP
  - More elegant and safer than C++
- **Go** (designed at Google - 2009)
  - Loosely based on C, but also quite different
  - Does not support traditional OOP
An Imperative-Based Object-Oriented Language: Java

• Developed at Sun in the early 1990s
  • C and C++ were not satisfactory for embedded electronic devices

• Based on C++
  • Significantly simplified (does not include `struct`, `union`, `enum`, pointer arithmetic, and half of the assignment coercions of C++)
  • Supports only OOP
  • Has references, but not pointers
  • Includes support for applets and a form of concurrency
Java Evaluation

- Eliminated many unsafe features of C++
- Supports concurrency
- Libraries for applets, GUIs, database access
- Portable: Java Virtual Machine concept, JIT compilers
- Widely used for Web programming
- Use increased faster than any previous language
- Most recent version, 7, released in 2011
Scripting Languages for the Web

• Perl
  • Designed by Larry Wall—first released in 1987
  • Variables are statically typed but implicitly declared
  • Three distinctive namespaces, denoted by the first character of a variable’s name
  • Powerful, but somewhat dangerous
  • Gained widespread use for CGI programming on the Web
  • Also used for a replacement for UNIX system administration language

• JavaScript
  • Began at Netscape, but later became a joint venture of Netscape and Sun Microsystems
  • A client-side HTML-embedded scripting language, often used to create dynamic HTML documents
  • Purely interpreted
  • Related to Java only through similar syntax

• PHP
  • PHP: Hypertext Preprocessor, designed by Rasmus Lerdorf
  • A server-side HTML-embedded scripting language, often used for form processing and database access through the Web
  • Purely interpreted
Scripting Languages for the Web

• Python
  • An OO interpreted scripting language
  • Type checked but dynamically typed
  • Used for CGI programming and form processing
  • Dynamically typed, but type checked
  • Supports lists, tuples, and hashes

• Ruby
  • Designed in Japan by Yukihiro Matsumoto (a.k.a, “Matz”)
  • Began as a replacement for Perl and Python
  • A pure object-oriented scripting language
    - All data are objects
  • Most operators are implemented as methods, which can be redefined by user code
  • Purely interpreted
Scripting Languages for the Web

- Lua
  - An OO interpreted scripting language
  - Type checked but dynamically typed
  - Used for CGI programming and form processing
  - Dynamically typed, but type checked
  - Supports lists, tuples, and hashes, all with its single data structure, the table
  - Easily extendable
Perl Example

$total = 0;

#the for loop gives $x the value of all the
#numbers from 1 to 100;
for ($x ( 1 .. 100 )
    $total += $x;  # again, the short form
}

print "The total from 1 to 100 is $total\n";

@flavors = ( "vanilla", "chocolate", "strawberry" );

for ( $flavor ( @flavors )
    print "We have $flavor milkshakes\n";
}

print "They are 2.95 each\n";
print "Please email your order for home delivery\n";
<HTML>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write("<h2>Table of Factorials</h2>" );
for(i = 1, fact = 1; i < 10; i++, fact *= i) {
    document.write(i + "! = " + fact);
    document.write("<br>");
}
</SCRIPT>
</BODY>
</HTML>
```php
$data = array(array('topics' => array('',
    'Syllabus',
    '<a href="notes/chapter1.pptx">Introduction</a>'),
    'reading' => array('Chapter 1'),
    array('topics' => array('<i>No Class - Holiday</i>','Introduction','Introduction'),
    'reading' => array('Chapter 1'),
    'graded' => array('Assignment 0 Due (+2%)', '', '')),
    array('topics' => array('',
    '<a href="notes/chapter2.pptx">Evolution of Programming Languages</a>'),
    'reading' => array('Chapter 2'),
    array('topics' => array('<a href="notes/lisp.pdf">Functional Programming Languages</a>')),
    'reading' => array('Chapter 15'),
    array('topics' => array('Syntax & Semantics','Chapter 3'),
    array('topics' => array('Lexical & Syntax Analysis','Chapter 4'),
    array('topics' => array('<i>No Class - Holiday (02.20)</i>','Names, Binding, Type Checking, Scopes', 'nbsp;'))),
```
while(list(&$d) = each($data))
{
  print "<tr>";
  printf("<td>%s<br>%s</td>",
    date($st == 6 ? "": "m.d", mktime(0,0,0,1,$st+=3,2012)),
    date("m.d", mktime(0,0,0,1,$st+=2,2012)),
    date("m.d", mktime(0,0,0,1,$st+=2,2012)));
}

while(list(&$k) = each($keys))
{
  print "<td>";
  if ($d[$k])
  {
    for ($i = 0; $i < count($d[$k]); $i++)
    {
      print $d[$k][$i];
      if ($i < count($d[$k]) - 1)
        print "<br>";
    }
  }
  print "</td>";
}
reset($keys);
print "</tr>";
}

<p><strong>Meeting Times</strong></p>

<ul type=disc>
  <li>Lecture: Monday, Wednesday and Friday 11:00 a.m. - 11:50 a.m. in EPS 108. </li>
</ul>

<p><strong>Textbook</strong></p>
Python Example

```python
# indent your Python code to put into an email
import glob
# glob supports Unix style pathname extensions
python_files = glob.glob('*.py')
for file_name in sorted(python_files):
    print '    {}' .format(file_name)

    with open(file_name) as f:
        for line in f:
            print '    ' + line.rstrip()

print
```
Ruby Example

class Person
  attr_reader :name, :age
  def initialize(name, age)
    @name, @age = name, age
  end
  def <=>(person) # Comparison operator for sorting
    @age <=> person.age
  end
  def to_s
    "#{@name (#{@age})"
  end
end

group = [
  Person.new("Bob", 33),
  Person.new("Chris", 16),
  Person.new("Ash", 23)
]

puts group.sort.reverse
The Flagship .NET Language: C#

• Part of the .NET development platform (2000)
• Based on C++, Java, and Delphi
• Includes pointers, delegates, properties, enumeration types, a limited kind of dynamic typing, and anonymous types
• Is evolving rapidly
C# Example

```csharp
using System;
namespace KodecsSharp.Example.Intro
{
    class AutoImplementedProperty
    {
        [SThread]
        static void Main(string[] args)
        {
            UserAccount user = new UserAccount(1);
            user.Username = "admin";
            user.FirstName = "System";
            user.LastName = "Administrator";

            Console.WriteLine(user);
        }
    }

    class UserAccount
    {
        //
        // Auto-implemented properties
        //
        public long Id { get; set; }
        public string Username { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

        /// <summary>
        /// Creates a new UserAccount with the specified Id.
        /// </summary>
        /// <param name="id">a UserAccount Id</param>
        public UserAccount(long id)
        {
            Id = id;
        }

        /// <summary>
        /// A string representation of this object.
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            return string.Format("Id: {0}; Username: {1}; " +
                                "FirstName: {2}; LastName: {3}"
                               , Id, Username, FirstName, LastName);
        }
    }
}
```
Summary

• Development, development environment, and evaluation of a number of important programming languages
• Perspective into current issues in language design