Programming Languages

Lecturer: William W.Y. Hsu
Class Objectives

• What are we going to do in this class?
  ▫ Compare and contrast different programming languages.

• What does this entail?
  ▫ Examine the way in which languages are designed and implemented.
Course Objectives

• Understand concepts in programming languages.
  ▫ Be able to explain common language constructs and features.
  ▫ Be able to explain the subtlety in language features such as scoping, binding, and parameter passing.
  ▫ Be able to simulate useful features in languages that lack them.

• Understand the general approach to implement a language.
  ▫ Be able to write a programming language translator (compiler/interpreter).

• Be able to program in *procedural, object-oriented, functional*, and *logical* programming languages.

• Ultimate goal: be able, in principle, to design a new programming language!
Course Topics

- **Language concepts.**
  - Classification of programming languages.
  - Common language constructs: sequencing, loops, conditions, etc.
  - Names, Scopes, and Bindings: How and when bindings for local names are defined in languages with scoping rules.
  - Control Flow: How programming constructs define control flow and how the choice of constructs can affect programming style.
  - Subroutines and Parameter Passing: How the subroutine calling mechanism is implemented and how and when parameters are passed and evaluated.
  - Exception Handling: How to improve the robustness of programs.
Course Topics

• Language implementations.
  ▫ Common techniques used in compilers and interpreters.
  ▫ Lexical analysis – identify correct words in a program.
  ▫ Syntax analysis – identify syntactically correct program structures.
  ▫ Semantics analysis – identify meaningful programs.

• Alternative programming
  ▫ Functional Programming: Programming with Scheme, F#.
  ▫ Logic Programming: Programming with Prolog.
What This Course is About?

• Some questions may have occurred to you at some points:
  ▫ Why C++ is designed as it is?
  ▫ What does it take to convert a C++ program into an executable?
  ▫ Are there any better programming language choices?
  ▫ What is a good programming language?
  ▫ .......

• This course gives answers to this type of questions.
Why Do This?

- For the **fun** of it!
- Understanding the basic principles makes it **easier to learn new languages**.
- Sometimes you need **different features of different languages**, and if you don’t know about other languages how can you use them?
- **More effectively utilize** the languages you already know.
- For example, if you need “fine-grained” control over system memory, then you **C++** would be better choice than **Java**. However, if you memory leaks are a big concern, then **Java** is a better choice than **C++**.
Ultra-brief History of Languages

- In the beginning, ENIAC (Electronic Numerical Integrator and Computer) programmers used patch cords.
- This gave them the raw power to compute trig tables.
Important Events in Programming Language History

- 1940s: The first electronic computers were monstrous contraptions.
  - Programmed in binary machine code by hand via switches and later by card readers and paper tape readers.
  - Code is not reusable or relocatable.
  - Computation and machine maintenance were difficult: machines had short mean-time to failure (MTTF) because vacuum tubes regularly burned out.
  - The term “bug” originated from a bug that reportedly roamed around in a machine causing short circuits.
Machine and Assembly Languages

• The next major revolution was **machine language**, which is just binary (or hexadecimal).
• Very quickly people realized that humans cannot write error free programs using just zeroes and ones without going insane.
• Hence, came **assembly language**, which uses human readable abbreviations to stand for machine code.
Machine Language Program

55 89 e5 53
83 ec 04 83
e4 f0 e8 31
00 00 00 89
.....

• Hard to maintain or write large programs
• Needs an easier way to write program
Assembly Languages

- **Assembly languages** were invented in the 50s’ to allow machine operations to be expressed in mnemonic abbreviations.
  - Enables larger, reusable, and relocatable programs.
  - Actual machine code is produced by an assembler.
  - Early assemblers had a one-to-one correspondence between assembly and machine instructions.

- “**Speedcoding**”: expansion of macros into multiple machine instructions to achieve a form of higher-level programming.
Assembly Language (example)

Start:

- lea A, a0
- lea B, a1
- lea C, a2
- clr.w d0
- clr.w d1
- clr.w d2
- add.w #5, d1
- add.w #6, d2
- move.w d1, (a0)
- move.w d2, (a1)
- add.w (a0), d0
- add.w (a1), d0
- move.w d0, (a2)
- jsr decout
- jsr newline
- jsr stop

Data:

- A: dc.w 1
- B: dc.w 1
- C: dc.w 1
Assembly Language Example

- Example MIPS assembly program to compute GCD.
- Example MIPS R4000 machine code of the assembly program

```
addiu sp,sp,-32
sw ra,20(sp)
jal getint
nop
jal getint
sw v0,28(sp)
lw a0,28(sp)
move v1,v0
beq a0,v0,D
slt at,v1,a0
A: beq at,zero,B
nop
b c
subu a0,a0,v1
B: subu v1,v1,a0
C: bne a0,v1,A
slt at,v1,a0
D: jal putint
nop
lw ra,20(sp)
addiu sp,sp,32
jr ra
move v0,zero
```

27bdfffd0 afbf0014 0c1002a8 00000000
0c1002a8 afa2001c 8fa4001c
00401825 10820008 0064082a 10200003
00000000 10000002 00832023
00641823 1483ffffa 0064082a 0c1002b2
00000000 8fbf0014 27bd0020
03e00008 00001025

Actual MIPS R4400 IC
Higher Level Languages

- Eventually, people realized that more complex programs are very difficult to write at the level of assembly language.
- So, eventually came higher level languages.

```java
class Test {
    public static void main(String args[]) {
        int A, B, C;
        A=5;
        B=6;
        C=A+B;
        System.out.print(C);
    }
}
```
The First High-Level Programming Language

- Mid 1950s: development of FORTRAN (FORmula TRANslator), the arguably first higher-level language.
  - Finally, programs could be developed that were machine independent!
- Main computing activity in the 50s: solve numerical problems in science and engineering.
- Other high-level languages soon followed:
  - Algol 58 was an improvement compared to Fortran.
  - COBOL for business computing.
  - Lisp for symbolic computing and artificial intelligence.
  - BASIC for "beginners".
  - C for systems programming.
FORTRAN 77 Example

PROGRAM GCD
C  variable names that start
C     with
C  I,J,K,L,N,M are
C     integers
C  read the parameters
READ (*, *) I, J
C  loop while I!=J
10 IF I .NE. J THEN
   IF I .GT. J THEN
      I = I - J
   ELSE
      J = J - I
   ENDIF
GOTO 10
ENDIF
C  write result
WRITE (*, *) 'GCD =', I
END

• FORTRAN is still widely used for scientific, engineering, and numerical problems, mainly because very good compilers exist.
• In the early days skeptics wrongly predicted that compilers could not beat hand-written machine code.
• FORTRAN 77 has:
  ▫ Subroutines, if-then-else, do-loops
  ▫ Types (primitive and arrays)
  ▫ Variable names are upper case and limited to 6 chars
  ▫ No recursion
  ▫ No structs/classes, unions
  ▫ No dynamic allocation
  ▫ No case-statements and no while-loops
FORTRAN I,II,IV,77

PROGRAM AVEX
INTEGER INTLST(99)
ISUM = 0
C read the length of the list
READ (*, *) LSTLEN
IF ((LSTLEN .GT. 0) .AND. (LSTLEN .LT. 100)) THEN
C read the input in an array
DO 100 ICTR = 1, LSTLEN
READ (*, *) INTLST(ICTR)
ISUM = ISUM + INTLST(ICTR)
100 CONTINUE
C compute the average
IAVE = ISUM / LSTLEN
C write the input values > average
DO 110 ICTR = 1, LSTLEN
IF (INTLST(ICTR) .GT. IAVE) THEN
WRITE (*, *) INTLST(ICTR)
END IF
110 CONTINUE
ELSE
WRITE (*, *) 'ERROR IN LIST LENGTH'
END IF
END

- FORTRAN had a dramatic impact on computing in early days
- Still used for numerical computation
FORTRAN 90,95,HPF

```
PROGRAM AVEX
    INTEGER INT_LIST(1:99)
    INTEGER LIST_LEN, COUNTER, AVERAGE

    C read the length of the list
    READ (*, *) LISTLEN
    IF ((LIST_LEN > 0) .AND. (LIST_LEN < 100)) THEN
        C read the input in an array
        DO COUNTER = 1, LIST_LEN
            READ (*, *) INT_LIST(COUNTER)
        END DO
        C compute the average
        AVERAGE = SUM(INT_LIST(1:LIST_LEN)) / LIST_LEN
        C write the input values > average
        DO COUNTER = 1, LIST_LEN
            IF (INT_LIST(COUNTER) > AVERAGE) THEN
                WRITE (*, *) INT_LIST(COUNTER)
            END IF
        END DO
    ELSE
        WRITE (*, *) 'ERROR IN LIST LENGTH'
    END IF
END
```

- Major revisions
  - Recursion
  - Pointers
  - Records
- New control constructs
  - `while-loop`
- Extensive set of array operations
- HPF (High-Performance Fortran) includes constructs for parallel computation
Declarative and Imperative programming

- There are two types of programming languages: declarative and imperative.
  - Declarative languages focus on **what** the computer should do.
  - Imperative languages focus on **how** the computer should do something.
Quicksort

- Quicksort sorts an array by recursively sorting “subarrays” as less than or greater than pivot values.
Quicksort in Haskell

If input is empty return empty.

Otherwise, return a list with all the values less than x both “qsort”ed and before x and all values greater than x both “qsort”ed and after x.

This junk defines lt_x as all values less than x, and ge_x as all values greater than or equal to x.

qsort [] = []
qsort (x:xs) = qsort lt_x ++ [x] ++ qsort ge_x

where
lt_x = [y | y <- xs, y < x]
ge_x = [y | y <- xs, y >= x]
Quicksort in C

Find the first element larger than the pivot value and the last element smaller than the pivot value.

If these values are on the “wrong side” of the pivot, swap them.

Repeat until no values are on the “wrong side.”

Swap the smallest value greater than or equal to the pivot with the pivot, which is at the end of the list.

Finally, recurse on the two sides.
QuickSort in Python

```python
import random

def quicksort(L):
    if len(L) > 1:
        pivot = random.randrange(len(L))
        elements = L[:pivot]+L[pivot+1:]
        left = [element for element in elements if element < L[pivot]]
        right = [element for element in elements if element >= L[pivot]]
        return quicksort(left)+[L[pivot]]+quicksort(right)
    return L
```

Dynamic typing system.

Randomize the pivot for average case.

Iterators within arrays.

The whole concept is functional, with writing with imperative constructs.
Quicksort in OCaml

```ocaml
let rec quicksort = function
  | [] -> []
  | x::xs -> let smaller, larger
    = List.partition(fun y -> y < x) xs
  in quicksort smaller @ (x::quicksort larger)
```
Quicksort in Prolog

quicksort([X|Xs],Ys) :-
    partition(Xs,X,Left,Right),
    quicksort(Left,Ls),
    quicksort(Right,Rs),
    append(Ls,[X|Rs],Ys).
quicksort([],[]).

partition([X|Xs],Y,[X|Ls],Rs) :-
    X \leq Y, partition(Xs,Y,Ls,Rs).
partition([X|Xs],Y,Ls,[X|Rs]) :-
    X > Y, partition(Xs,Y,Ls,Rs).
partition([],Y,[],[]).

append([],Ys,Ys).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).
Quicksort Comparison

- Notice how much more complex this program is in C (an imperative language) than Haskell (a declarative language).
- However, without a very good compiler, the quicksort in C will likely run faster than in Haskell!
Types of Languages

- **Von Neumann** languages allow for computation by focusing on manipulating data elements.
- **Object-oriented** languages allow for computation by modeling principles as a series of semi-independent “objects”.
- **Scripting** languages are a subset of von Neumann languages and are serve as “glue” between more robust languages in order to facilitate rapid development.
Types of Languages

- **Functional** languages are based on functions and recursion.
- **Dataflow** languages focus on the flow of information between nodes.
- **Logic** languages model programs as a series of logical statements.
Lisp

(DEFINE (avex lis)
  (filtergreater lis (/ (sum lis) (length lis))))
)
(DEFINE (sum lis)
  (COND
    ((NULL? lis) 0)
    (ELSE (+ (CAR lis) (sum (CDR lis)))))
  )
)
(DEFINE (filtergreater lis num)
  (COND
    ((NULL? lis) '())
    ((> (CAR lis) num) (CONS (CAR lis)
      (filtergreater (CDR lis) num)))
    (ELSE (filtergreater (CDR lis) num))
  )
)

- Lisp (LIst Processing)
- The original functional language developed by McCarthy as a realization of Church's lambda calculus
- Many dialects exist, including Common Lisp and Scheme
- Very powerful for symbolic computation with lists
- Implicit memory management with garbage collection
- Influenced functional programming languages (ML, Miranda, Haskell)
Algol 60

The original block-structured language
- Local variables in a statement block
- First use of Backus-Naur Form (BNF) to formally define language grammar
- All subsequent imperative programming languages are based on it
- Not widely used in the US
- Unsuccessful successor Algol 68 is large and relatively complex

code

comment a vex program
begin
    integer array intlist [1:99];
    integer listlen, counter, sum, average;
    sum := 0;
    comment read the length of the input list
    readint (listlen);
    if (listlen > 0) L (listlen < 100) then
        begin
            comment read the input into an array
            for counter := 1 step 1 until listlen do
                begin
                    readint (intlist[counter]);
                    sum := sum + intlist[counter]
                end;
            comment compute the average
            average := sum / listlen;
            comment write the input values > average
            for counter := 1 step 1 until listlen do
                if intlist[counter] > average then
                    printint (intlist[counter])
            end
        else
            printstring ("Error in input list length")
        end
end
COBOL

IDENTIFICATION DIVISION.
PROGRAM-ID.  EXAMPLE.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-370.

DATA DIVISION.
WORKING-STORAGE SECTION.
77 FAHR  PICTURE 999.
77 CENT  PICTURE 999.

PROCEDURE DIVISION.
DISPLAY 'Enter Fahrenheit ' UPON CONSOLE.
ACCEPT FAHR FROM CONSOLE.
DISPLAY 'Celsius is ' CENT UPON CONSOLE.
GOBACK.

- Originally developed by the Department of Defense.
- Intended for business data processing.
- Extensive numerical formatting features and decimal number storage.
- Introduced the concept of records and nested selection statement.
- Programs organized in divisions:
    IDENTIFICATION: Program identification
    ENVIRONMENT: Types of computers used
    DATA: Buffers, constants, work areas
    PROCEDURE: The processing parts (program logic).
BASIC

REM avex program
DIM intlist(99)
sum = 0
REM read the length of the input list
INPUT listlen
IF listlen > 0 AND listlen < 100 THEN
REM read the input into an array
FOR counter = 1 TO listlen
    INPUT intlist(counter)
    sum = sum + intlist(counter)
NEXT counter
REM compute the average
average = sum / listlen
REM write the input values > average
FOR counter = 1 TO listlen
    IF intlist(counter) > average THEN
        PRINT intlist(counter);
    NEXT counter
ELSE
    PRINT "Error in input list length"
END IF
END

- BASIC (Beginner’s All-Purpose Symbolic Instruction Code)
- Intended for interactive use (interpreted) and easy for "beginners"
- Goals: easy to learn and use for non-science students
- Structure of early basic dialects were similar to Fortran
- Classic Basic
- QuickBasic (see example)
- MS Visual Basic is a popular dialect
PL/I

AVEX: PROCEDURE OPTIONS (MAIN);
    DECLARE INTLIST (1:99) FIXED;
    DECLARE (LISTLEN, COUNTER, SUM, AVERAGE) FIXED;
    SUM = 0;
    /* read the input list length */
    GET LIST (LISTLEN);
    IF (LISTLEN > 0) & (LISTLEN < 100) THEN
        DO;
            /* read the input into an array */
            DO COUNTER = 1 TO LISTLEN;
                GET LIST (INTLIST(COUNTER));
                SUM = SUM + INTLIST(COUNTER);
            END;
            /* compute the average */
            AVERAGE = SUM / LISTLEN;
            /* write the input values > average */
            DO COUNTER = 1 TO LISTLEN;
                IF INTLIST(COUNTER) > AVERAGE THEN
                    PUT LIST (INTLIST(COUNTER));
            END;
        ELSE
            PUT SKIP LIST ('ERROR IN INPUT LIST LENGTH');
        END;
    END AVEX;

• Developed by IBM
  ▫ Intended to replace FORTRAN, COBOL, and Algol
• Introduced exception handling
• First language with pointer data type
• Poorly designed, too large, too complex
Ada and Ada95

• Originally intended to be the standard language for all software commissioned by the US Department of Defense
• Very large
• Elaborate support for packages, exception handling, generic program units, concurrency
• Ada 95 is a revision developed under government contract by a team at Intermetrics, Inc.
  ▫ Adds objects, shared-memory synchronization, and several other features

with TEXT_IO;
use TEXT_IO;
procedure AVEX is
  package INT_IO is new INTEGER_IO (INTEGER);
  use INT_IO;
  type INT_LIST_TYPE is array (1..99) of INTEGER;
  INT_LIST : INT_LIST_TYPE;
  LIST_LEN, SUM, AVERAGE : INTEGER;
begin
  SUM := 0;
  -- read the length of the input list
  GET (LIST_LEN);
  if (LIST_LEN > 0) and (LIST_LEN < 100) then
    -- read the input into an array
    for COUNTER := 1 .. LIST_LEN loop
      GET (INT_LIST(COUNTER));
      SUM := SUM + INT_LIST(COUNTER);
    end loop;
    -- compute the average
    AVERAGE := SUM / LIST_LEN;
    -- write the input values > average
    for counter := 1 .. LIST_LEN loop
      if (INT_LIST(COUNTER) > AVERAGE) then
        PUT (INT_LIST(COUNTER));
        NEW_LINE;
      end if
    end loop;
  else
    PUT_LINE ("Error in input list length");
  end if;
end AVEX;
Smalltalk-80

Developed by XEROX PARC:
first IDE with windows-based graphical user interfaces (GUIs)
The first full implementation of an object-oriented language

```smalltalk
class name  Avex
superclass  Object
instance variable names  intlist

"Class methods"
"Create an instance"
new
  ^ super new

"Instance methods"
"Initialize"
initialize
  intlist <- Array new: 0

"Add int to list"
add int to list:
  add: n | oldintlist |
    oldintlist <- intlist.
    intlist <- Array new: intlist size + 1.
    intlist <- replaceFrom: 1 to: intlist size with: oldintlist.
    ^ intlist at: intlist size put: n

"Calculate average"
average | sum |
  sum <- 0.
  1 to: intlist size do:
    [:index | sum <- sum + intlist at: index].
  ^ sum // intlist size

"Filter greater than average"
filtergreater: n | oldintlist i |
  oldintlist <- intlist.
  i <- 1.
  1 to: oldintlist size do:
    [:index | (oldintlist at: index) > n
      ifTrue: [oldintlist at: i put: (oldintlist at: index)]]
  intlist <- Array new: oldintlist size.
  intlist replaceFrom: 1 to: oldintlist size with: oldintlist
```
The most widely used logic programming language
Declarative: states what you want, not how to get it
Based on formal logic
Pascal

program avex(input, output);

  type
    intlisttype = array [1..99] of integer;
  var
    intlist : intlisttype;
    listlen, counter, sum, average : integer;
begin
  sum := 0;
  (* read the length of the input list *)
  readln(listlen);
  if ((listlen > 0) and (listlen < 100)) then
    begin
      (* read the input into an array *)
      for counter := 1 to listlen do
        begin
          readln(intlist[counter]);
          sum := sum + intlist[counter]
        end;
      (* compute the average *)
      average := sum / listlen;
      (* write the input values > average *)
      for counter := 1 to listlen do
        if (intlist[counter] > average) then
          writeln(intlist[counter])
    end
    else
      writeln('Error in input list length')
  end.

- Designed by Swiss professor Niklaus Wirth
- Designed for teaching "structured programming"
- Small and simple
- Had a strong influence on subsequent high-level languages Ada, ML, Modula
C (ANSI C, K&R C)

- One of the most successful programming languages
- Primarily designed for systems programming but more broadly used
- Powerful set of operators, but weak type checking and no dynamic semantic checks

```c
main()
{ int intlist[99], listlen, counter, sum, average;
  sum = 0;
  /* read the length of the list */
  scanf("%d", &listlen);
  if (listlen > 0 && listlen < 100)
  { /* read the input into an array */
    for (counter = 0; counter < listlen; counter++)
    { scanf("%d", &intlist[counter]);
      sum += intlist[counter];
    }
    /* compute the average */
    average = sum / listlen;
    /* write the input values > average */
    for (counter = 0; counter < listlen; counter++)
    if (intlist[counter] > average)
      printf("%d\n", intlist[counter]);
  }
  else
    printf("Error in input list length\n");
}
```
C++

main()
{
    std::vector<int> intlist;
    int listlen;
    /* read the length of the list */
    std::cin >> listlen;
    if (listlen > 0 && listlen < 100)
    {
        int sum = 0;
        /* read the input into an STL vector */
        for (int counter = 0; counter < listlen; counter++)
        {
            int value;
            std::cin >> value;
            intlist.push_back(value);
            sum += value;
        }
        /* compute the average */
        int average = sum / listlen;
        /* write the input values > average */
        for (std::vector<int>::const_iterator it = intlist.begin();
             it != intlist.end();
             ++it)
        {
            if ((*it) > average)
                std::cout << (*it) << std::endl;
        }
        else
            std::cerr << "Error in input list length" << std::endl;
    }
    else
        std::cerr << "Error in input list length" << std::endl;
}
Java

import java.io;

class Avex {
    public static void main(String args[]) throws IOException {
        DataInputStream in = new DataInputStream(System.in);
        int listlen, counter, sum = 0, average;
        int [] intlist = int[100];
        // read the length of the list
        listlen = Integer.parseInt(in.readLine());
        if (listlen > 0 && listlen < 100) {
            // read the input into an array
            for (counter = 0; counter < listlen; counter++) {
                intlist[counter] = Integer.valueOf(in.readline()).intValue();
                sum += intlist[counter];
            }
            // compute the average
            average = sum / listlen;
            // write the input values > average
            for (counter = 0; counter < listlen; counter++) {
                if (intlist[counter] > average)
                    System.out.println(intlist[counter] + "\n");
            }
        } else
            System.out.println("Error in input length\n");
    }
}
Other Notable Languages

• **C#**
  - Similar to Java, but platform dependent (MS .NET)
  - Common Language Runtime (CLR) manages objects that can be shared among the different languages in .NET

• **Simula 67**
  - Based on Algol 60
  - Primarily designed for discrete-event simulation
  - Introduced concept of coroutines and the class concept for data abstraction

• **APL**
  - Intended for interactive use ("throw-away" programming)
  - Highly expressive functional language makes programs short, but hard to read

• **Scripting languages**
  - Perl, Python, Ruby, …
Why are There so Many Programming Languages?

• Evolution
  ▫ Design considerations: What is a good or bad programming construct?
  ▫ Early 70s: structured programming in which goto-based control flow was replaced by high-level constructs (e.g. while loops and case statements).
  ▫ Late 80s: nested block structure gave way to object-oriented structures.
Why are There so Many Programming Languages?

• Special Purposes
  ▫ Many languages were designed for a specific problem domain, e.g:
    • Scientific applications
    • Business applications
    • Artificial intelligence
    • Systems programming
    • Internet programming

• Personal Preference
  ▫ The strength and variety of personal preference makes it unlikely that anyone will ever develop a universally accepted programming language.
What Makes a Programming Language Successful?

- **Expressive Power**
  - Theoretically, all languages are equally powerful (Turing complete)
  - Language features have a huge impact on the programmer's ability to read, write, maintain, and analyze programs
  - Abstraction facilities enhance expressive power

- **Ease of Use for Novice**
  - Low learning curve and often interpreted, e.g. Basic and Logo

- **Ease of Implementation**
  - Runs on virtually everything, e.g. Basic, Pascal, and Java
What Makes a Programming Language Successful?

- Open Source
  - Freely available, e.g. Java

- Excellent Compilers and Tools
  - Fortran has extremely good compilers
  - Supporting tools to help the programmer manage very large projects

- Economics, Patronage, and Inertia
  - Powerful sponsor: Cobol, PL/I, Ada
  - Some languages remain widely used long after "better" alternatives
Important to Know PL by Trend

TIOBE Programming Community Index

Source: www.tiobe.com
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>Java</td>
<td>19.010%</td>
<td>-0.26%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>C</td>
<td>11.303%</td>
<td>-3.43%</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>C++</td>
<td>5.800%</td>
<td>-1.94%</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td>C#</td>
<td>4.907%</td>
<td>+0.07%</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td>Python</td>
<td>4.404%</td>
<td>+0.34%</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>^</td>
<td>PHP</td>
<td>3.173%</td>
<td>+0.44%</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>^</td>
<td>JavaScript</td>
<td>2.705%</td>
<td>+0.54%</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
<td>Visual Basic .NET</td>
<td>2.518%</td>
<td>-0.19%</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>^</td>
<td>Perl</td>
<td>2.511%</td>
<td>+0.39%</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>^</td>
<td>Assembly language</td>
<td>2.364%</td>
<td>+0.60%</td>
</tr>
<tr>
<td>11</td>
<td>14</td>
<td>^</td>
<td>Delphi/Object Pascal</td>
<td>2.278%</td>
<td>+0.87%</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>^</td>
<td>Ruby</td>
<td>2.278%</td>
<td>+0.85%</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>\downarrow</td>
<td>Visual Basic</td>
<td>2.045%</td>
<td>+0.25%</td>
</tr>
<tr>
<td>14</td>
<td>17</td>
<td>^</td>
<td>Swift</td>
<td>1.963%</td>
<td>+0.80%</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>\downarrow</td>
<td>Objective-C</td>
<td>1.884%</td>
<td>-1.31%</td>
</tr>
<tr>
<td>16</td>
<td>37</td>
<td>^</td>
<td>Groovy</td>
<td>1.637%</td>
<td>+1.27%</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>^</td>
<td>R</td>
<td>1.605%</td>
<td>+0.60%</td>
</tr>
<tr>
<td>18</td>
<td>15</td>
<td>\downarrow</td>
<td>MATLAB</td>
<td>1.538%</td>
<td>+0.31%</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td></td>
<td>PL/SQL</td>
<td>1.349%</td>
<td>+0.21%</td>
</tr>
<tr>
<td>20</td>
<td>95</td>
<td>^</td>
<td>Go</td>
<td>1.270%</td>
<td>+1.10%</td>
</tr>
<tr>
<td>Position</td>
<td>Programming Language</td>
<td>Ratings</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>SAS</td>
<td>1.111%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Scratch</td>
<td>1.049%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>ARAP</td>
<td>1.024%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>D</td>
<td>1.022%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>COBOL</td>
<td>0.983%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Dart</td>
<td>0.972%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Transact-SQL</td>
<td>0.773%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Fortran</td>
<td>0.750%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Lua</td>
<td>0.733%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Ada</td>
<td>0.655%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Lisp</td>
<td>0.629%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Scala</td>
<td>0.572%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Prolog</td>
<td>0.484%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Scheme</td>
<td>0.432%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>LabVIEW</td>
<td>0.400%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Logo</td>
<td>0.400%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>RPG (OS/400)</td>
<td>0.347%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Haskell</td>
<td>0.284%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Erlang</td>
<td>0.287%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Apex</td>
<td>0.262%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Ladder Logic</td>
<td>0.262%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Rust</td>
<td>0.255%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Bash</td>
<td>0.232%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Q</td>
<td>0.231%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>F#</td>
<td>0.219%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Awk</td>
<td>0.204%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>VHDL</td>
<td>0.197%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Alice</td>
<td>0.191%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>VBScript</td>
<td>0.177%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>C shell</td>
<td>0.171%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Next 50 Programming Languages

- The following list of languages denotes #51 to #100. Since the differences are relatively small, the programming languages are only listed (in alphabetical order).

(Visual) FoxPro, ABC, ActionScript, Algol, APL, AutoLISP, bc, Boo, Bourne shell, CFML, CL (OS/400), Clojure, Common Lisp, Crystal, Curl, Elixir, Elm, Factor, Forth, Hack, Icon, IDL, Inform, Io, J, Julia, Korn shell, Maple, Mathematica, ML, Modula-2, MQL4, MS-DOS batch, NATURAL, NXT-G, OCaml, OpenCL, Oz, Pascal, PL/I, PostScript, PowerShell, REXX, Smalltalk, SPARK, SPSS, Standard ML, Stata, Tcl, Verilog
<table>
<thead>
<tr>
<th>Sep 2015</th>
<th>Sep 2014</th>
<th>Change</th>
<th>Programming Language</th>
<th>Ratings</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>▶️</td>
<td>Java</td>
<td>19.565%</td>
<td>+5.43%</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>▼️</td>
<td>C</td>
<td>15.621%</td>
<td>-1.10%</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>▶️</td>
<td>C++</td>
<td>6.782%</td>
<td>+2.11%</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>▶️</td>
<td>C#</td>
<td>4.909%</td>
<td>+0.56%</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>▶️</td>
<td>Python</td>
<td>3.664%</td>
<td>+0.88%</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>▶️</td>
<td>PHP</td>
<td>2.530%</td>
<td>-0.59%</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>▶️</td>
<td>JavaScript</td>
<td>2.342%</td>
<td>-0.11%</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>▶️</td>
<td>Visual Basic .NET</td>
<td>2.062%</td>
<td>+0.53%</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>▶️</td>
<td>Perl</td>
<td>1.899%</td>
<td>+0.53%</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>▼️</td>
<td>Objective-C</td>
<td>1.821%</td>
<td>-8.11%</td>
</tr>
<tr>
<td>11</td>
<td>29</td>
<td>▶️</td>
<td>Assembly language</td>
<td>1.806%</td>
<td>+1.22%</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>▶️</td>
<td>Ruby</td>
<td>1.783%</td>
<td>+0.50%</td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>▶️</td>
<td>Delphi/Object Pascal</td>
<td>1.745%</td>
<td>+0.59%</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td></td>
<td>Visual Basic</td>
<td>1.532%</td>
<td>+0.26%</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>▶️</td>
<td>Pascal</td>
<td>1.298%</td>
<td>+0.40%</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>▶️</td>
<td>Swift</td>
<td>1.188%</td>
<td>+0.34%</td>
</tr>
<tr>
<td>17</td>
<td>19</td>
<td>▶️</td>
<td>MATLAB</td>
<td>1.181%</td>
<td>+0.36%</td>
</tr>
<tr>
<td>18</td>
<td>20</td>
<td>▶️</td>
<td>PL/SQL</td>
<td>1.082%</td>
<td>+0.27%</td>
</tr>
<tr>
<td>19</td>
<td>21</td>
<td>▶️</td>
<td>R</td>
<td>1.045%</td>
<td>+0.24%</td>
</tr>
<tr>
<td>20</td>
<td>31</td>
<td>▶️</td>
<td>COBOL</td>
<td>0.994%</td>
<td>+0.42%</td>
</tr>
<tr>
<td>Position</td>
<td>Programming Language</td>
<td>Ratings</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>SAS</td>
<td>0.974%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Fortran</td>
<td>0.888%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Ada</td>
<td>0.792%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Stata</td>
<td>0.788%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>D</td>
<td>0.768%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>ASAP</td>
<td>0.732%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Scala</td>
<td>0.731%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Dart</td>
<td>0.702%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Lisp</td>
<td>0.694%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>OpenEdge ABL</td>
<td>0.635%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Lua</td>
<td>0.598%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>F#</td>
<td>0.590%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Prolog</td>
<td>0.510%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Groovy</td>
<td>0.505%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Logo</td>
<td>0.499%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Transact-SQL</td>
<td>0.421%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Erlang</td>
<td>0.398%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Haskell</td>
<td>0.349%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>RPC (OS/400)</td>
<td>0.325%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>LabVIEW</td>
<td>0.324%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Forth</td>
<td>0.291%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>(Visual) FoxPro</td>
<td>0.291%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Scheme</td>
<td>0.256%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Go</td>
<td>0.254%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>ActionScript</td>
<td>0.218%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Awk</td>
<td>0.218%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>ML</td>
<td>0.213%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>VHDL</td>
<td>0.209%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>Ladder Logic</td>
<td>0.206%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>Rust</td>
<td>0.108%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Important to Know PL by Trend
### Ratings Aug 2013 vs Delta Aug 2012

<table>
<thead>
<tr>
<th>Category</th>
<th>Ratings Aug 2013</th>
<th>Delta Aug 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object-Oriented Languages</td>
<td>57.8%</td>
<td>+0.6%</td>
</tr>
<tr>
<td>Procedural Languages</td>
<td>35.7%</td>
<td>-2.0%</td>
</tr>
<tr>
<td>Functional Languages</td>
<td>4.2%</td>
<td>+0.8%</td>
</tr>
<tr>
<td>Logical Languages</td>
<td>2.3%</td>
<td>+0.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Ratings Aug 2013</th>
<th>Delta Aug 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statically Typed Languages</td>
<td>69.3%</td>
<td>-2.4%</td>
</tr>
<tr>
<td>Dynamically Typed Languages</td>
<td>30.7%</td>
<td>+2.4%</td>
</tr>
</tbody>
</table>
RedMonk Q316 Programming Language Rankings

Popularity Rank on Stack Overflow (by # of Tags)

Popularity Rank on GitHub (by # of Projects)
History of Computer Programming Languages
History of Computer Programming Languages
History of Computer Programming Languages
History of Computer Programming Languages

2000

- C#
- Pointer Arithmetic
- Garbage Collection
- Java
- VB
- Python
- Ruby
Periodic Table of Programming Languages

<table>
<thead>
<tr>
<th>Periodic Table of Programming Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualisation of the evolution of popular programming languages</td>
</tr>
</tbody>
</table>

Each row represents roughly a decade, starting on the second row with the 1920's up to the 2000's on the final row. The first row is pre-1950 with the two mechanical programming systems from which all others have evolved - the first from around 1837 created by Charles Babbage and Ada Lovelace.

The colours denote the programming paradigm that the language in question originally supported or the primary paradigm for which it is known. Some languages may have evolved to support other paradigms over time which are not shown.
The Evolution Of Computer Programming Languages

Hex  Assembler  C  Fortran  C++  Java  Ruby
Class Policy

- Ask questions anytime!
- You may leave anytime for WC without asking.
  - It’s human rights.
- Refrain from using any messaging devices.
- Phones to vibration.
- Eating and drinking is ALLOWed, however…
  - If anyone leaves trash behind, this policy will be canceled. Penalty will be applied from this point on.
- Name calling will not be conducted. It is your duty to make up yourself if you skip classes.
  - No need to ask for leave.
  - However, no make up exams will be given! (Unless you have a good reason)
- Refrain from sleeping in classes.
  - Get sleep in your bed for maximum quality.
  - If it happens too much, I will start to deduct point adjustments of final scores.
- Refrain from chatting too loud of personal matters.
  - I will remove you from the classroom because you are interfering with the class.
Resources

- Computation resources – Linux clusters @ ACL.
- Machine locations:
  WatchTower.cse.ntou.edu.tw (140.121.197.19)
  Fortress.cse.ntou.edu.tw (140.121.197.20)
  WarMill.cse.ntou.edu.tw (140.121.197.21)
  Barracks.cse.ntou.edu.tw (140.121.197.22)
- Quota 1G.
- Can login with ssh softwares.
  - putty
  - Bitvise ssh client
Course Info

- Textbook:

- Grading (tentative):
  - 40% Homework assignments. (4-5 assignments)
    - DKP system for undetermined plagiarized works.
  - 30% Midterm examination.
  - 30% Final examination.
Course Info

- Website:
  - [http://capitol.cse.ntou.edu.tw/wwyhsu/?page_id=1297](http://capitol.cse.ntou.edu.tw/wwyhsu/?page_id=1297)

- TA:
  - 王浩勳 [wangha81@gmail.com](mailto:wangha81@gmail.com) (Primary TA)
  - 游旻叡 [yu830425@gmail.com](mailto:yu830425@gmail.com) (Secondary TA)