Introduction to Programming Languages

COMP 141

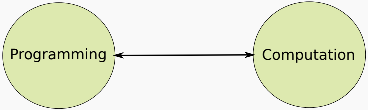
Sepehr Amir-Mohammadian
Dept. of Computer Science
University of the Pacific

What are we going to learn today?

- The origin of programming languages
- Paradigms of programming

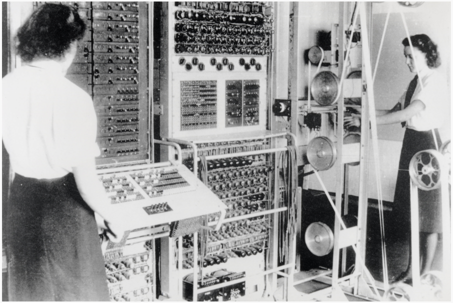
Why Programming Languages?

- *Programming Language*: a mechanism to communicate with the computer, expressing what that computer needs to accomplish



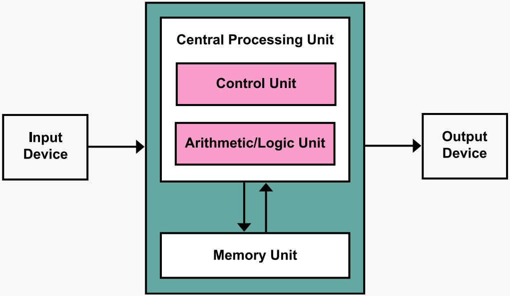
- The abstract notion of *computation* and how to *program* mutually affect each other.
- *Knowing the principles of programming languages $\rightarrow$ understanding the way(s) to attack a problem*
- *Theory of Programming Languages*: a fundamental part of computer science

Origin of PLs



- Up to mid *1940s*:
  - Operators had to turn off and rewire the computer for each computation using switches
  - Communicating the intended computation by *hardware reconfiguration*

Von Neumann Architecture



- **John von Neumann** proposed a new architecture for computers in *1945*
  - *Fixed hardware* with a small set of general purpose operations
  - *Input channel* to receive the user input in *binary format* that specify
    - the operations to use
    - the operands to those operations
  - *Output channel* to return the computation result

Von Neumann Architecture (cont.)

```
0010001000000100
0010010000000100
0001011001000010
0011011000000011
1111000000100101
0000000000000101
0000000000000110
0000000000000000
```

Machine Code for Little Computer 3 (LC-3)

Welcome to *machine language*!

- Operators had to enter the *binary user input* to the memory using a set of switches
  - ... so operators became the first programmers!

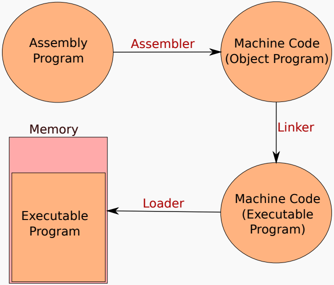
Machine Language Shortcomings

- Machine language programming was *tedious and error prone*. Why?
- There was pretty huge *gap* between
  - what programmer had in mind as the solution, and
  - what the computer understood as the corresponding program.
- *Solution*: Let's use *mnemonic symbols* for instructions and memory locations!
  - For example, rather than 0010001000000100 we may say `LD R1 ARG`.
  - Called *Assembly* language!
  - introduced in early *1950s*.

Assembly 8086 Example Code

```
org 100h

mov ax, 5        ; set ax to 5.
mov bx, 2        ; set bx to 2.

jmp calc         ; go to 'calc'.

back: jmp stop   ; go to 'stop'.

calc:
add ax, bx       ; add bx to ax.
jmp back         ; go 'back'.

stop:

ret              ; return to operating system.
```

Assembly Language: A Bigger Picture

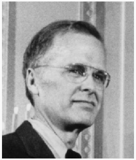


- *Assembler*: translates assembly code to machine code
- *Object program*: machine code that is not usually executable
- *Linker*: combines multiple object programs into a single executable program
- *Executable program*: machine code that can be executed by the processor
- *Loader*: Loads the machine code into memory

Assembly Language Shortcomings

- No abstraction for conventional mathematical notations
  - For example, some algebraic expression like $(x \times 3) - (y + z)$
  - Requires translating such abstractions to machine-dependent notations
  - ... So still the *gap* exists.
- Platform dependent
  - Each type of computer hardware architecture has its own machine language instruction set and requires its own dialect of assembly language

Next Step: FORTRAN



- FORTRAN: FORmula TRANslation
- Developed by **John Backus** in the late *1950s*
- Originally designed for an *IBM* machine
- Introduced *algebraic expressions* and *floating point numbers*
- Shortcomings:
  - Originally lacked the *structured control statements* and data structures of later high-level languages
  - To some degree had similarities to Assembly: *platform-dependent*
- It has undergone several major *revisions* through its history, supports many features that others languages do

FORTRAN II Example Code

```
C AREA OF A TRIANGLE - HERON'S FORMULA
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT -
C INTEGER VARIABLES START WITH I,J,K,L,M OR N
      READ(5,501) IA,IB,IC
  501 FORMAT(3I5)
      IF (IA) 701, 777, 701
  701 IF (IB) 702, 777, 702
  702 IF (IC) 703, 777, 703
  777 STOP 1
  703 S = (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - IA) * (S - IB) * (S - IC) )
      WRITE(6,801) IA,IB,IC,AREA
  801 FORMAT(4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     $13H SQUARE UNITS)
      STOP
      END
```

ALGOL comes in

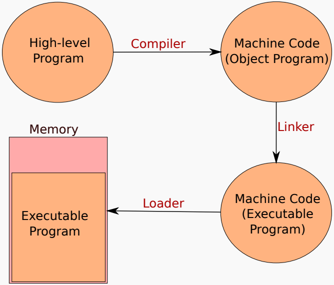
- ALGOL: ALGOrithmic Language, released in *1960*
- Known as *ALGOL-60*
- Provided a *standard notation* for computer scientists to publish algorithms in journals
- Included *structured control statements* for
  - *sequencing* (`begin-end` blocks),
  - *loops* (`for` loop), and
  - *selection* (`if` and `if-else` statements)
- Supported *different numeric types*
- Introduced the *array* structure
- Supported *procedures*, including *recursive procedures*
- First language to receive a *formal specification*

ALGOL-60 Example Code

```
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 copied 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
```

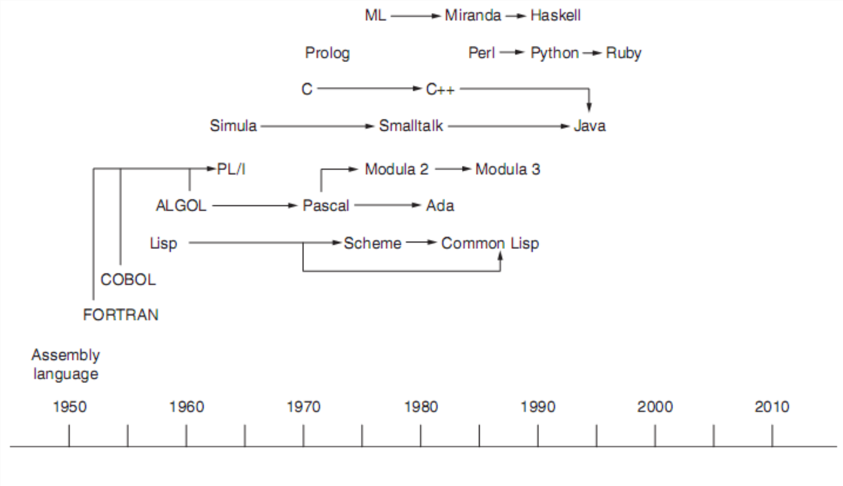
ALGOL: Platform-Independence

- ALGOL is machine architecture independent. How?
- It has a *compiler* for each architecture.
- *Compiler*: A program that translates high level programs into machine code.



- Many other PLs have descended from ALGOL, including *Pascal (1970)* and *Ada (1980)*.

Timeline of PLs



Programming Paradigms

Programming paradigms are a way to *categorize* languages.

- *Imperative* programming
- *Functional* programming
- *Logic* programming
- *Object-oriented* programming

These paradigms could be *orthogonal* to each other!

Imperative Programming

- An imperative language has the following properties
  - *Sequential execution* of instructions
  - Use of *variables representing memory locations*
  - Use of *assignment* to change the values of variables
- Majority of PLs are imperative
  - FORTRAN, ALGOL, Pascal, Ada, C, C++, Java, Perl, Python, etc.

Example: Factorial in C

```
int factorial (int n) {
  int i, fact = 1;
  for (i = 1; i <= n; i++)
    fact = fact * i;
  return fact;
}
```

Functional Programming

- An functional language is based on the *abstract notion of function*.
- Functional programming is based on a core calculus named *λ-calculus*.
- λ-calculus is developed by mathematician **Alonzo Church** in *1930s* using the *theory of recursive functions*.
- Well-known functional languages:
  - Lisp, Scheme, ML, Miranda, Haskell, Coq, Agda, Idris, etc.

Example: Factorial in Haskell

```
factorial 0 = 1
factorial n = n * factorial(n-1)
```

Logic Programming

- Logic programming is based on *symbolic logic*.
- A program consists of *a set of logical rules and facts*.
- Datalog and Prolog are among the well-known logic-based languages.

Example: Factorial in Prolog

```
factorial(0,1).
factorial(N,F) :-
  N>0, N1 is N-1, factorial(N1,F1), F is N * F1.
```

Object-Oriented Programming (OOP)

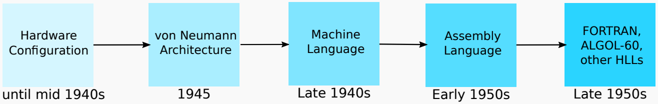
- In OOP, *reusable code* operates in a way to mimic behaviors of *real-world objects*.
- Java, C++, C#, Python, PHP, Ruby, Perl, Objective-C, Swift, and Scala are among the well-known object-oriented languages.

Example: Factorial in Java

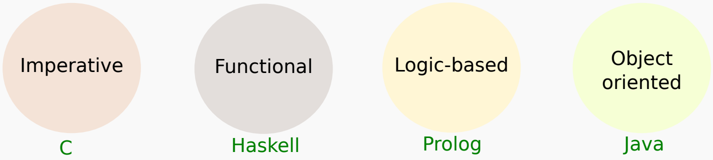
```
public BigInteger factorial(int n) {
  BigInteger fact = new BigInteger("1");
  for (int i = 1; i <= n; i++) {
    fact = fact.multiply(new BigInteger(i + ""));
  }
  return fact;
}
```

What we learned today?

- We studied the origin of PLs



- We introduced different programming paradigms



Next Session

- Some basic definitions in PLs
- The future of PLs
- Different design criteria in PLs