History of Programming Languages
“If you don't know history, then you don't know anything. You are a leaf that doesn't know it is part of a tree.”

Michael Crichton
Some attributes of a good programming language

• **Clarity** – You can write the algorithm as you envision it in the application domain

• **Readability** – Someone reading a program will understand what it does

• **Intuitiveness** – It works the way you expect

• **Scalability** – The ability to write large or small programs in the language
More Attributes

- **Safety** – The language minimizes the chance to make subtle errors
- **Efficiency** – Programs written in the language execute fast and efficiently
- **Large library** – Many functions are available
- **Support for abstraction** – Allows encapsulating concepts
Charles Babbage

Charles Babbage built a mechanical computer starting in 1822. He never completed the machine.
Ada Lovelace

Augusta Ada, Countess of Lovelace, was the daughter of Lord Byron and friend of Charles Babbage

She is considered the first computer programmer
Analog Computers

• For many years electrical engineers have used mathematics to model circuits
• If you want to solve a mathematical equation, you can wire a circuit whose solution is the equation
• Analog computers are not digital, they represent a number by the voltage on a wire
• There is no programming language for an analog computer
ENIAC

- **Electronic Numerical Integrator And Computer**
- John Eckert and J. Presper Mauchly
- University of Pennsylvania
- Trajectory tables for weapons
- Started 1943
- Finished 1946
  - Too late for war effort
- Used until 1955
ENIAC - details

• Programmed manually by switches and wiring
• Decimal (*not binary*)
• 20 registers of 10 digits
• 18,000 vacuum tubes
• 30 tons
• 15,000 square feet
• 140 kW power
• 5,000 additions/sec
von Neumann Architecture

- Stored Program concept
- Main memory holds programs and data
- ALU operating on binary data
- Control unit interpreting instructions from memory and executing
- Input and output equipment operated by control unit
- Completed 1952
Machine Language

• Computers execute machine language
• All other computer languages are translated to machine language for execution by a compiler
• Machine language is binary and very tedious to use
• Machine language is unique to each architecture
Machine Language Format for Imaginary Computer

Opcodes

<table>
<thead>
<tr>
<th>Add</th>
<th>Subtract</th>
<th>Multiply</th>
<th>Divide</th>
<th>Load</th>
<th>Store</th>
<th>JumpEql</th>
<th>Jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Instruction Formats

Load and Store

<table>
<thead>
<tr>
<th>opcode</th>
<th>register</th>
<th>index register</th>
<th>memory address</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>21</td>
</tr>
</tbody>
</table>

Add, Sub, Mult and Divide

<table>
<thead>
<tr>
<th>opcode</th>
<th>unused</th>
<th>reg 1</th>
<th>reg 2</th>
<th>reg 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

```
1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 1 0 0 0 0 1 0 0 0 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 0 0 0 0 0
1 0 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```
The image contains a sample instruction with its corresponding opcode and operation descriptions. The instruction is encoded in binary and can be decoded as follows:

**Opcode**: 100 = 4 = Load instruction

**Register**: 0001 = R1

**Index**: 0110 = R6

**Address**: 0001000 = 8

The instruction is Load R1, 8[R6].
Program to calculate a square root

better = number;
do {
    good = better;
    better = (better + number/ better) / 2;
}while (good!= better);
<table>
<thead>
<tr>
<th>addr</th>
<th>machine</th>
<th>assembler</th>
</tr>
</thead>
<tbody>
<tr>
<td>004a</td>
<td>8b 45 e0</td>
<td>mov eax, number</td>
</tr>
<tr>
<td>004d</td>
<td>89 45 f8</td>
<td>mov good, eax</td>
</tr>
<tr>
<td>0050</td>
<td>89 45 ec</td>
<td>mov better, eax</td>
</tr>
<tr>
<td>0053</td>
<td>8b 45 ec</td>
<td>mov eax, better again:</td>
</tr>
<tr>
<td>0056</td>
<td>89 45 f8</td>
<td>mov good, eax</td>
</tr>
<tr>
<td>0059</td>
<td>8b 45 e0</td>
<td>mov eax, number</td>
</tr>
<tr>
<td>005c</td>
<td>ba 00 00 00 00</td>
<td>mov edx, 0</td>
</tr>
<tr>
<td>0061</td>
<td>f7 7d f8</td>
<td>idiv good</td>
</tr>
<tr>
<td>0064</td>
<td>03 45 f8</td>
<td>add eax, good</td>
</tr>
<tr>
<td>0067</td>
<td>d1 f8</td>
<td>sar eax, 1</td>
</tr>
<tr>
<td>0069</td>
<td>89 45 ec</td>
<td>mov better, eax</td>
</tr>
<tr>
<td>006c</td>
<td>3b 45 f8</td>
<td>cmp eax, good</td>
</tr>
<tr>
<td>006f</td>
<td>75 e2</td>
<td>jne SHORT again</td>
</tr>
</tbody>
</table>
Assembly language

• Assembly language is a simplified means of writing machine language
• Addresses are represented by names
• Opcodes are represented by mnemonics
• Each line of assembly language generally creates one machine language instruction
Assemblers

• An assembler is a program that translates assembly language to machine language
• It converts the mnemonics to the proper numerical opcode
• The assembler calculates all of the numerical addresses
• Assemblers read one line at a time and create the machine language for that assembler statement
Forward and Backward References

• Consider the following simple assembler program

  `bkwd  mov  eax, forwd  // move forwd to eax
         add  eax, 5     // add 5 to eax
         jmp  bkwd      // jump to bkwd

  forwd  DW  47       // data word`

• When the assembler reads the first line, it does not know the address for forwd

• When it gets to the third line, it knows the address of bkwd
Two Pass Assembler

• A two pass assembler reads the assembler source code and calculates addresses.
• It creates a symbol table that has all of the program names, such as `bkwd` and `forwd`, and their address.
• It then reads the source a second time and generates the output using the symbol table to get addresses.
• The assembler needs only enough data space to hold the symbol table.
One Pass Assembler

• If you can fit the generated machine code in memory, only one pass over the assembler source is needed
• When a forward reference is found, the location where the address is needed is kept in the symbol table
• When the symbol is found, its value is placed in the needed locations
Assembler Macros

• Many assemblers have a macro feature that allows the program to define new assembler source code statements

\[
\text{swap} \quad \text{define macro}
\]

\[
\begin{align*}
\text{mov} & \text{ eax, } \$1 \\
\text{mov} & \text{ ebx, } \$2 \\
\text{mov} & \text{} \$2, \text{ eax} \\
\text{mov} & \text{} \$1, \text{ ebx}
\end{align*}
\]

\[
\text{end macro}
\]

\[
\text{swap} \quad \text{dog, cat}
\]

\[
\begin{align*}
\text{mov} & \text{} \text{ eax, dog} \\
\text{mov} & \text{} \text{ ebx, cat} \\
\text{mov} & \text{} \text{ cat, eax} \\
\text{mov} & \text{} \text{ dog, ebx}
\end{align*}
\]
First Compiler

- Alick Glennie developed Autocode in 1952
- It ran on the Manchester Mark 1

- The Manchester Mark 1 was a prototype for the Ferranti Mark 1, the world's first commercially available general-purpose electronic computer in 1951
Autocode Example

- Calculates $f(t) = \sqrt{|t|} + 5t^3$

```plaintext
INTEGERS +5 →c  # Put 5 into c
→t          # Load argument from lower accumulator to variable t
  +tTESTA Z  # Put |t| into lower accumulator
  -t
ENTRY Z     # Run square root subroutine on lower accumulator

SUBROUTINE 6 →z # value and put the result into z

  +tt→y→x     # Calculate t^3 and put it into x
  +tx→y→x
+z+c→x       # Put z + (c * x) into lower accumulator and return
```

example from Wikipedia
Formula Translating System - Fortran

• Created at IBM by John Backus in 1957
• It reduced the number of programming statements necessary by a factor of 20
• Still in use, primarily for high performance computing
Fortran Facts

• Spaces were ignored and could be added or removed
• \texttt{INABOX} and \texttt{IN A BOX} are the same variable
• Lines could be given a number
  
  23 \hspace{5pt} \texttt{IF (DOG .EQ. CAT) GOTO 47}

• Variables that began with I through N were INTEGER and all others were REAL unless declared otherwise
• Supports complex numbers
• Arrays start at one
• Exponentiation in equations: \texttt{CAT ** 3} is \texttt{CAT}^{3}
Many Fortran Versions

• Fortran has evolved over the years.
• If statement formats

```
IF (CAT - DOG) 34, 789, 46
IF (CAT .EQ. DOG) GO TO 789
IF (CAT .EQ. DOG) THEN ... ELSE ... END IF
```
How good is Fortran?

• **Clarity** – Allows you to write equations and engineering algorithms

• **Readability** – Generally OK, although the jumps make for spaghetti code

• **Intuitiveness** – It generally works the way you expect

• **Scalability** – Some very large computationally intensive programs are written in Fortran
More Fortran Attributes

• **Safety** – Does not check for subtle errors or security

• **Efficiency** – Fortran’s simple memory model and no pointers allows compilers to create efficient code

• **Large library** – Many math functions are available

• **Support for abstraction** – Does not provide for objects or other types of abstraction
Common Business-Oriented Language - Cobol

• Designed in 1959 for the Department of Defense (DoD)
• Grace Hopper is commonly referred to as “the (grand)mother of COBOL”
• Intended to create a portable programming language for data processing
• Still in use
• When she recommended that a new programming language be developed using English words, she “was told very quickly that [she] couldn't do this because computers didn't understand English.”
Hello World in Cobol

IDENTIFICATION DIVISION.
PROGRAM-ID. 'HELLO'.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. GNULINUX.
OBJECT-COMPUTER. HERCULES.
SPECIAL-NAMES.
       CONSOLE IS CONSL.
DATA DIVISION.
PROCEDURE DIVISION.
00-MAIN.
       DISPLAY 'HELLO, WORLD' UPON CONSL.
STOP RUN.
Cobol Considerations

• Excessively verbose

\[
\text{cat} = \text{dog} + 1;
\]

Java or C++

add one to dog giving cat.

Cobol

• No local variables

• Good support for record data types

• Sorting is a built in language feature
How good is Cobol?

• **Clarity** – Designed to look like English.
• **Readability** – Can be read to some extent by a non-programmer
• **Intuitiveness** – Easier for non-programmers than professional programmers
• **Scalability** – There are no methods. Everything is one big program
More Cobol Attributes

- **Safety** – Some data type protection
- **Efficiency** – Pretty good for report writing
- **Large library** – Small built in library. Sorting and merging are part of the language
- **Support for abstraction** – Terrible! No methods or objects
APL readability

• Find all prime numbers from 1 to R
  \( (~R \in R \circ \times R) / R \leftarrow 1 \downarrow tR \)

• Conway’s game of life

  \( \text{life} \leftarrow \{ \uparrow 1 \omega \vee . \land 3 4 = +/, \neg 1 0 1 \circ . \ominus -1 0 1 \circ . \ominus \subset \omega \} \)
Early Programming Languages

• 1952 – Autocode
• 1957 – FORTRAN
• 1958 – LISP
• 1958 – ALGOL
• 1959 – COBOL
• 1962 – APL
• 1962 – SNOBOL
• 1964 – BASIC
• 1964 – PL/I
Programming Paradigm Development

• Simula (late 1960s) was the first language designed to support **object-oriented** programming
• Smalltalk (mid-1970s) provided a complete ground-up design of an **object-oriented** language
• Prolog (1972) was the first **logic** programming language
• ML (1973) was one of the first statically typed **functional** programming languages
Reading for Friday

• Read chapter 1 of the textbook
• Read