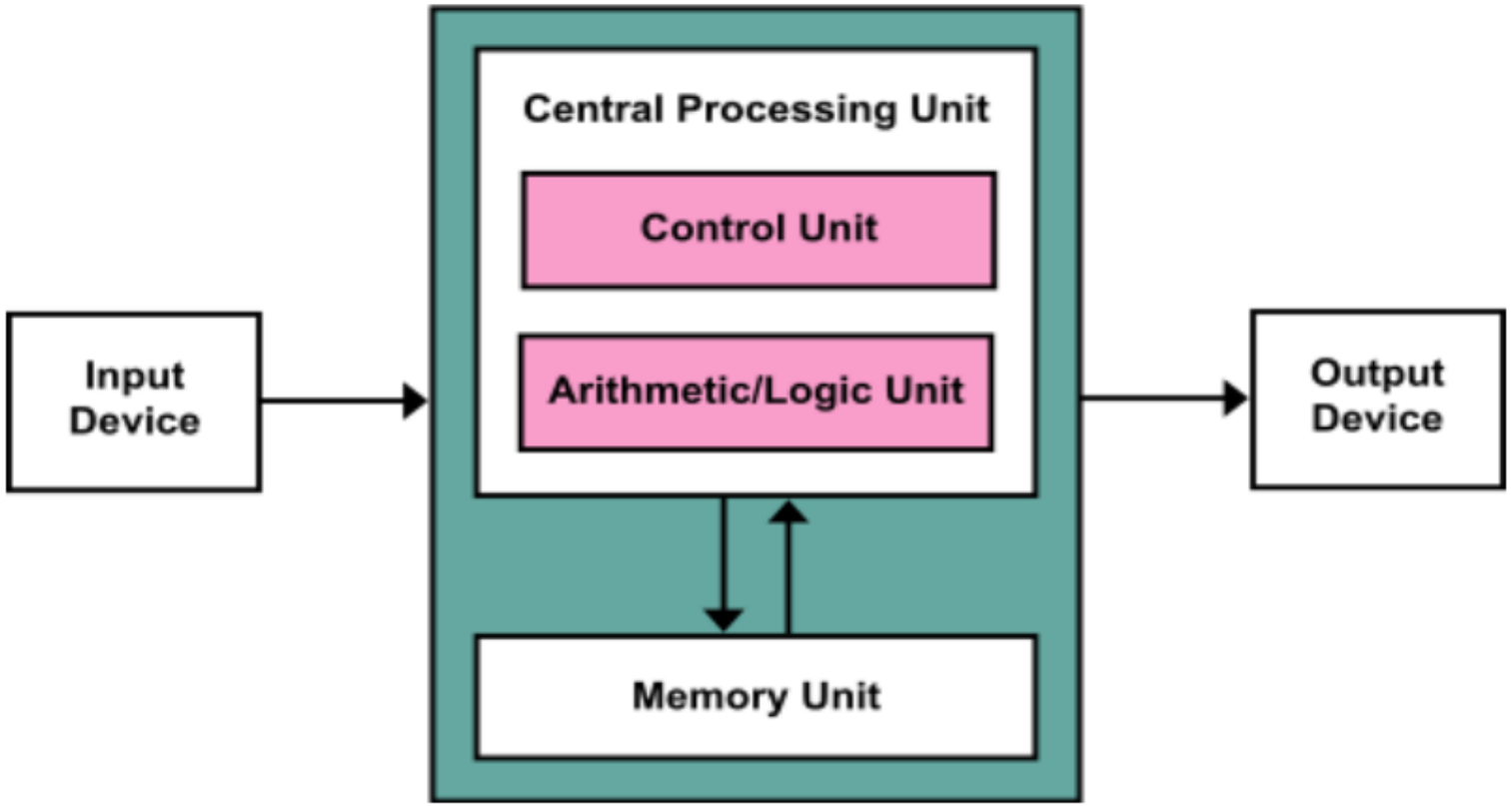
Evolution of Programming Languages: From Machine Language to Python

Dr. Bart Stuck

24 March 2024

What Makes Up a Computer: von Neumann Model



Types of Computer Applications: Example

- A. Scientific Computing:
  - Scientific computing is a field of computing that focuses on developing algorithms and software for solving complex mathematical problems.
  - It is used in scientific research, engineering, and other fields that require high-performance computing.
  - Supercomputers are often used for simulations, data analysis, and complex computations
- B. Embedded Systems:
  - Embedded Systems are computer systems that are designed to perform specific functions within a larger system or machine.
  - They are used in a wide range of applications, from consumer electronics to industrial automation.
  - Embedded systems are often designed to operate without human interaction and have real-time computing constraints
- C. Commercial Computing
  - Commercial Computing refers to the use of computers in business and commerce.
  - It includes a wide range of applications, from data processing and resource management to office productivity tasks.
  - Commercial computing is used in a variety of industries, including finance, healthcare, and retail.

Global Computing Market Size by Segment

- Scientific Computing
  - The Global High-Performance Computing Market size is forecasted to reach **USD 50.3 Billion** by the year 2028 and is expected to grow exhibiting a **Compound Annual Growth Rate (CAGR) of 6.3%**
- Embedded Systems
  - The global market for embedded systems is projected to reach **USD 163.2 billion** by 2031 with a **CAGR of 6.5%** from 2022 to 2031
- Commercial Computing
  - The global market size for banking and finance computing applications was estimated to be **USD 146.65 billion** in 2024 and is expected to reach **USD 271.75 billion by 2029**

Scientific Computing

- Seymour Cray

- Burton Smith

Computer Language Usage by Segment

- Scientific Computing
  - Popular programming languages used in Scientific Computing include C, C++, Java, and Python
  - Recently, researchers at MIT have developed a new programming language called A Tensor Language (ATL), which is specifically designed for high-performance computing. (Image processing, Deep-Learning Applications)
- Embedded Systems
  - According to a survey by IEEE Spectrum, the top two most popular and used programming languages in embedded systems are C and C++
- Commercial Computing
  - The most widely used commercial programming computer language is COBOL.
  - COBOL is a compiled English-like computer programming language designed for business use.
  - It is imperative, procedural and, since 2002, object-oriented.
  - COBOL is a programming language that reads like regular English and is most commonly used for business and administrative purposes

Top 5 Largest Programs Ever Written

- Human Genome Project:
  - Human Genome Project is a scientific research project with the aim to determine the sequence of human DNA.
  - The ultimate goal is to map all the genes of the human genome. It is already the largest collaborative project in biology till date.
  - The project started in 1990 and ended in 2003. The project was performed across twenty different universities.
  - The software for analyzing the human genome and map the nucleotide base pairs of DNA took **3300 billion lines of code**. In fact, the coding took more time than the actual execution of the project.
  - Human Genome Project | The McDonnell Genome Institute | Washington University in St. Louis (wustl.edu)
  - Human Epigenome Project
- Google:
  - Google is the largest platform in terms of the internet services it provides.
  - This includes the Google search engine which is the largest part of the Google platform, Gmail, Google Drive, Google Calendar, Google+, Google Translate, Google Photos, Google Notes, Google Maps, etc.
  - All these constitute nearly **2000 billion lines of code**.
  - With the constant upgrading and addition of features and new services, in the next few decades, Google is going to overcome the Human Genome Project in terms of lines of codes.

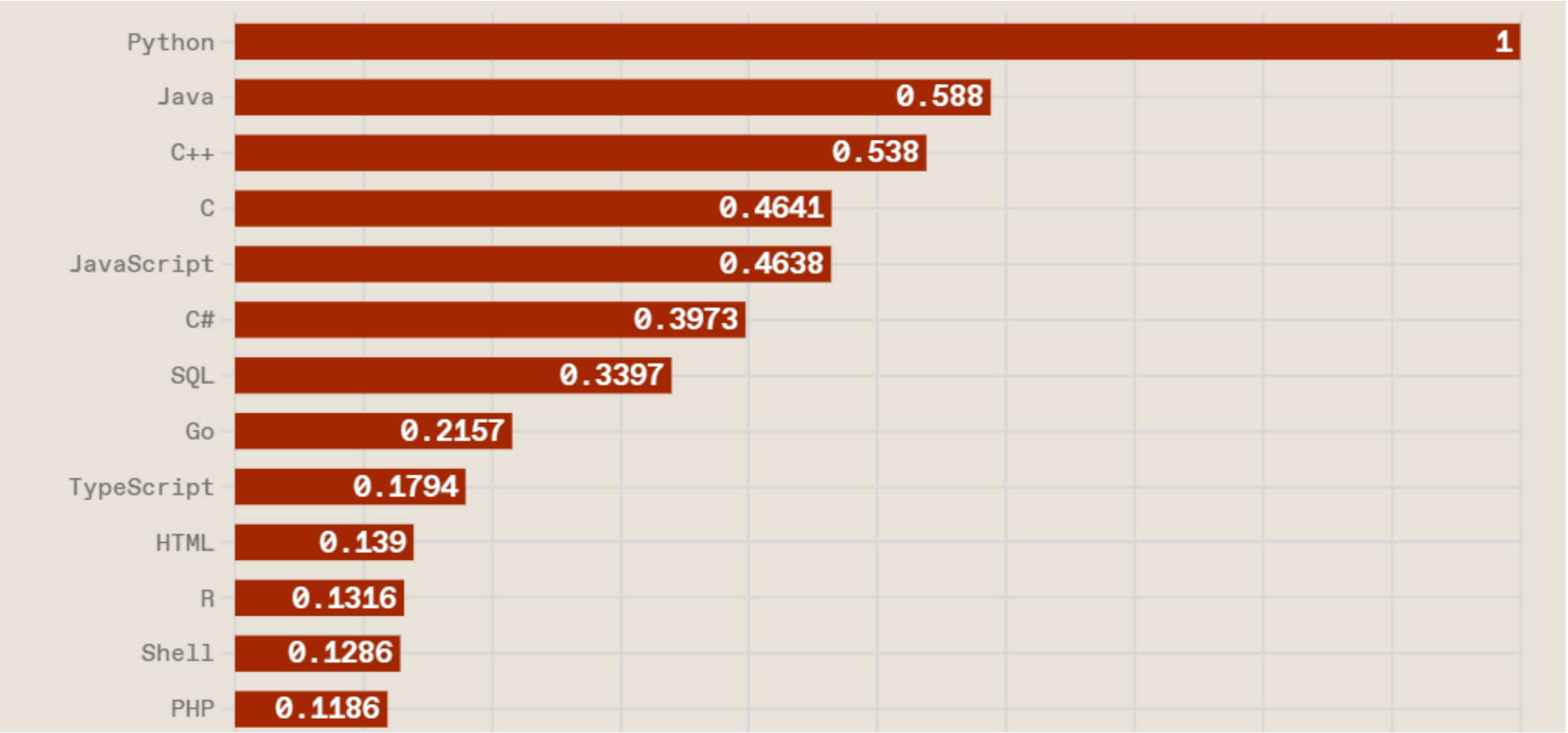
Top 5 Largest Programs Written (continued)

- Average High-End Car Software:
  - This should come as a shock to many. A lot of people have the opinion that the operating systems have the most lines of codes which is not far from the truth.
  - The car software in itself is an operating system and more. It is like the Windows with all the inbuilt software applications.
  - It is important to note that not all the car software that you see in the dashboard has millions of lines of code. We all know how robust the dashboard of high-end cars like BMW, Mercedes and likewise is.
  - The higher price you pay, the better the dashboard software will be. In fact, with the introduction of automation and driver-less driving, the lines of code for the car software have increased exponentially.
  - It has been found that average high-end car software has **over 100 million lines of code** to execute everything perfectly and provide the buyer with a value for their money.
  - C++ is the most popular language

Top 5 Largest Programs Written (continued)

- Mac OS X:
  - Mac OS X is the latest operating system from Apple, and it is well-received by the Apple fans who were tired of lack of innovation and user-friendliness in the previous Mac OS versions.
  - It is truly a major development from Apple as it combines the desktop and mobile devices under one platform.
  - Its competitors like Google and Microsoft are still struggling in this department.
  - Mac OS X has cross-platform functionality with iOS 8 and above. Mac OS X is considered to be the largest operating system ever written.
  - It contains **over 85 million lines of codes**
- Facebook:
  - After Google, Facebook is hands down the largest online platform.
  - This includes Facebook social network platform, the Facebook Messenger, gaming and other apps.
  - The one thing common about Google and Facebook as online platforms is that they are continuously upgrading and adding new features.
  - Facebook with all its services has **over 60 million lines of code.**

Top Programming Languages in 2023 (IEEE)



What Language Does a Computer Use To Do Actual Computing?

- A Computer Uses Machine Language in Binary Format
- For Ease of Programming, Early Computers Used Assembly Language
  - It is a symbolic representation of machine code instructions that can be easily understood by humans
  - Assembly code is converted into executable machine code by a utility program referred to as an **Assembler**
  - **It is relatively time intensive vs high level languages for developing robust software modules**
- A high-level programming language is a programming language that enables development of a program in a user-friendly and abstract way, without having to deal with the details of the computer's hardware or processor.
  - Examples of High Level Languages: Fortran, COBOL, Java, Python, C, C++
  - A Program called Compiler translates high-level language statements to machine code.

Example of a Computer: Apollo Guidance Computer AGC

- The **Apollo Guidance Computer** (**AGC**) was a digital computer produced for the Apollo program that was installed on board each Apollo command module (CM) and Apollo Lunar Module (LM).
- The AGC provided computation and electronic interfaces for guidance, navigation, and control of the spacecraft.The AGC was the first computer based on silicon integrated circuits.
- The computer's performance was comparable to the first generation of home computers from the late 1970s, such as the Apple II, TRS-80, and Commodore PET.

Apollo Guidance Computer AGC

- The AGC has a 16-bit word length, with 15 data bits and one parity bit. Most of the software on the AGC is stored in a special read-only memory known as core rope memory, fashioned by weaving wires through and around magnetic cores, though a small amount of read/write core memory is available.
- Astronauts communicated with the AGC using a numeric display and keyboard called the DSKY (for "display and keyboard", pronounced "DIS-kee"). The AGC and its DSKY user interface were developed in the early 1960s for the Apollo program by the MIT Instrumentation Laboratory and first flew in 1966.

Apollo Guidance Computer Instruction Set

- The instruction format used 3 bits for opcode, and 12 bits for address. Block I had 11 instructions: TC, CCS, INDEX, XCH, CS, TS, AD, and MASK (basic), and SU, MP, and DV (extra).
- The first eight, called basic instructions, were directly accessed by the 3-bit op. code.
- The final three were denoted as extracode instructions because they were accessed by performing a special type of TC instruction (called EXTEND) immediately before the instruction.
- Total number of instructions=33
- Total storage=2,048 words erasable magnetic storage + 36,864 words read only core rope memory (16 bit word: 15 bits data, 1 bit odd parity)=77.824 kilobytes total storage

Examples of AGC Instruction Set

- **AD k (Add)**
  - The "Add" instruction adds the contents of a memory location “k” into the accumulator.
- **BZF k (Jump)**
  - The "Branch Zero to Fixed" instruction jumps to a memory location “k”
- **CA k (Load)**
  - The "Clear and Add" instruction moves the contents of a memory location “k” into the accumulator
- **COM (Complement)**
  - Complements the Contents of Accumulator
- **LXCH k (Exchange)**
  - Exchanges the value in the L register with a value stored in memory location “k”
- **NOOP**
  - No operation
- **CCS (Count, Compare, Skip)**
- **INDEX**
  - Add the data retrieved at the address specified by the instruction to the next instruction.

Example of a Simple Program Using AGC Instruction Set

- Problem:
  - Sum of first N integers starting with one (1)
- Problem in AGC Assembly Language
- 10 1 #INITIAL#
- 20 1 #SUM#
- 30 N #NUMBER OF INTEGERS#
- 40 CA 10
- 50 ADD 10
- 60 CA 10
- 70 CCS
- 80 INDEX
- 5 distinct instructions

Using Chat GPT to Write the Same Program Using AGC Instruction set

- # Extend the next instruction DCA INDEX using EXTEND
- # Clear and Add to the Accumulator, using INDEX as the location. This sets the accumulator to 0. INDEX TEN
- # Set the INDEX register to the value at location TEN (10). CA ZERO
- # Clear the Accumulator and then add the value at ZERO (0). LOOP ADS INDEX
- # Add to Storage. Adds the contents of the accumulator to the value at location INDEX and stores the result back in INDEX. ADS TOTAL
- # Add the contents of INDEX to TOTAL, storing the result in TOTAL. SU ONE
- # Subtract the value at location ONE (1) from the accumulator. CCS A
- # Count, Compare, and Skip. If the Accumulator is negative, jump to the next instruction. TCF LOOP
- # Transfer Control to LOOP if the above CCS doesn't jump. CS TOTAL
- # Clear and Subtract. This effectively loads the negative of TOTAL into the accumulator. TC Q
- # Transfer Control to Quit (or the next part of your program). ONE OCT 00001
- # Define ONE as octal 1 ZERO OCT 00000 # Define ZERO as octal 0 TEN OCT 00012
- # Define TEN as octal 10 (decimal 10) INDEX ERASE
- # Allocate an erasable memory location for INDEX TOTAL ERASE
- # Allocate an erasable memory location for TOTAL

Same Problem In Python

- Problem:
  - Sum of first N integers starting with one (1)
- Problem in Python programming language
- Def sum(n)
- a, b = 0, 1
- While a<n:
- Print (a, end=‘ ‘)
- a,b=b,1+b
- Print()
- 5 distinct lines of code total

Chat GPT Response to Question
Python Program to Add All Integers From 1 to 10

- Here is a Python program that adds the integers starting from 1 and ending at 10:
- ```python
- # Initialize the sum variable total = 0
- # Loop through the integers from 1 to 10 and add them to the total for i in range(1, 11): total += i
- # Print the total sum print("The sum of integers from 1 to 10 is:", total) ```
- You can run this program in a Python environment to see the sum of integers from 1 to 10.

How Does a Compiler Work?

- A compiler is a program that translates code written in a programming language into another language.
- The main steps of a compiler are:
  - Lexical analysis: the compiler breaks the source code into tokens that represent the individual elements of the program.
  - Syntactic and semantic analysis: the compiler checks the syntax and meaning of the code and builds an intermediate representation of the program.
  - Optimization: the compiler improves the performance and efficiency of the intermediate representation.
  - Output code generation: the compiler produces the final code in the target language.

How Does Lexical Analysis Work?

- Lexical analysis is the first phase of a compiler, also known as scanning
- The process of lexical analysis can be broken down into the following stages:
  - **Input preprocessing**: This stage involves cleaning up the input text and preparing it for lexical analysis. This may include removing comments, whitespace, and other non-essential characters from the input text.
  - **Tokenization**: This is the process of breaking the input text into a sequence of tokens. This is usually done by matching the characters in the input text against a set of patterns or regular expressions that define the different types of tokens.
  - **Token classification**: In this stage, the lexer determines the type of each token. For example, in a programming language, the lexer might classify keywords, identifiers, operators, and punctuation symbols as separate token types.
  - **Token validation**: In this stage, the lexer checks that each token is valid according to the rules of the programming language. For example, it might check that a variable name is a valid identifier, or that an operator has the correct syntax.
  - **Output generation**: In this final stage, the lexer generates the output of the lexical analysis process, which is typically a list of tokens. This list of tokens can then be passed to the next stage of compilation or interpretation.

Syntactic Analysis & Semantic Analysis

- Syntactic Analysis Functions:
  - Functions of Syntactic Analysis (also known as parsing) is the process of analyzing the input code to determine its structure and ensure that it conforms to the rules of the language's grammar.
  - The output of this phase is an abstract syntax tree (AST), which represents the structure of the input code in a tree-like format
- Semantic Analysis Functions:
  - Type Checking: Ensures that data types are used in a way consistent with their definition.
  - Label Checking: A program should contain label references.
  - Flow Control Check: Keeps a check that control structures are used in a proper manner. (example: no break statement outside a loop)

Optimization and Code Generation

- Optimization:
  - The optimization process is generally implemented using a sequence of optimizing transformations, algorithms which take a program and transform it to produce a semantically equivalent output program that uses fewer resources or executes faster
  - Some examples of optimization techniques include peephole optimizations, local optimizations, and global optimizations
- Code Generation
  - The input to the code generator typically consists of a parse tree or an abstract syntax tree. The tree is converted into a linear sequence of instructions, usually in an intermediate language such as three-address code
  - This is followed by:
    - Instruction selection: which instructions to use.
    - Instruction scheduling: in which order to put those instructions.
    - Register allocation: the allocation of variables to processor registers.

Number of Machine Language Statements For a Java Statement

- The number of machine language instructions that result from compiling a typical Java statement depends on the complexity of the statement.
- A single high-level language statement may lead to several assembly (machine) language instructions
- For example, the following Java statement a=b+c-d; corresponds to four instructions: LOAD B, ADD C, SUBTRACT D, and STORE A

What Is An Interpreter?

- An interpreter is a computer program that executes instructions written in a high-level language.
- It is used to directly execute program instructions written using one of the many high-level programming languages.
- Interpreters enable other programs to run on a computer or server.
- They process program code at run time, checking the code for errors line by line.

Which Programming Languages Are Based on Interpreters?

- Programming languages that are based on interpreters include Python, Ruby, Perl, PHP, and MATLAB.
- In an interpreted language, the source code is not directly translated by the target machine.
- Instead, a different program, aka the interpreter, reads and executes the code.

Compilers vs. Interpreters

- A compiler and an interpreter are two types of software that convert high-level programming languages into machine code that computers can understand.
- The main difference between the two is that a compiler translates the entire program into machine code before running it, while an interpreter translates the program line by line as it runs
- Programs that are compiled into native machine code tend to be faster than interpreted code.
- This is because the process of translating code at run time adds to the overhead, and can cause the program to be slower overall.

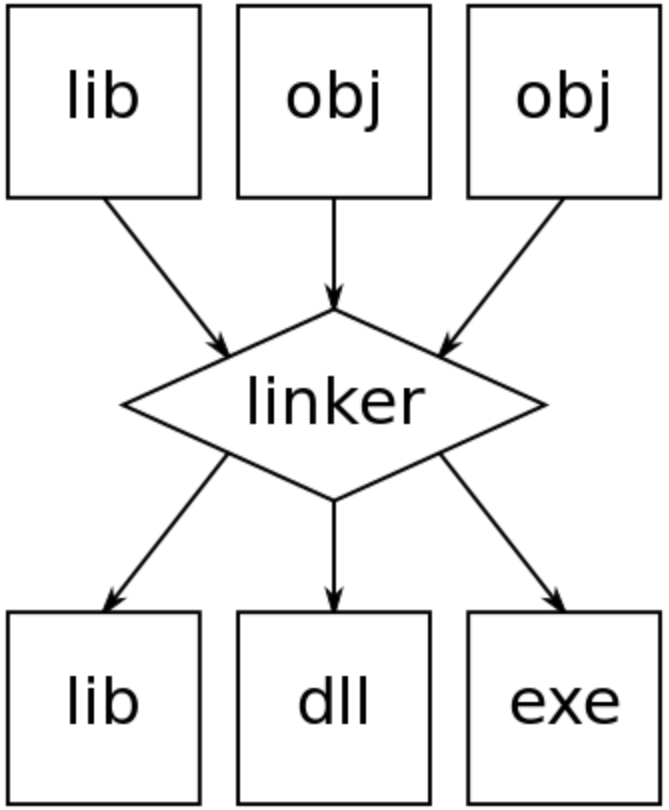
Compilers vs. Interpreters (cont.)

Compiler	Interpreter
Translates the entire program into machine code before running it	Translates the program line by line as it runs
Takes a long time to analyze the source code	Takes less time to analyze the source code
Generates object code which requires more memory	Does not generate object code, hence is memory efficient
Programming languages like C, C++, Java use compilers	Programming languages like JavaScript, Python, Ruby use interpreters

An Illustration of the Linking Process

While compilers (and assemblers) generally produce machine code directly executable by computer hardware, they can often (optionally) produce an intermediate form called object code. This is basically the same machine specific code but augmented with a symbol table with names and tags to make executable blocks (or modules) identifiable and relocatable.

Compiled programs will typically use building blocks (functions) kept in a library of such object code modules. A linker is used to combine (pre-made) library files with the object file(s) of the application to form a single executable file. The object files that are used to generate an executable file are thus often produced at different times, and sometimes even by different languages (capable of generating the same object format).



How Many Instructions Does A Computer Need to Solve Typical Computation Problem?

- Answer: LOAD, STORE, JUMP, ADD and SUBTRACT
- Using AGC: XCH(LOAD), TC(JUMP), TS(TRANSFER TO STORAGE), AD(ADD), SU(SUBTRACT)
- For improved effectiveness, most computers have additional instructions
- For instance, AGC had 33 instructions when it could use far less
- IBM 360 had more than 140 instructions with all variations
- DEC PDP11 had 138 instructions

How Many Instructions Does a Computer Need to Be Useful?

- The first FORTRAN compiler for the IBM 360 was developed by IBM in 1964.
- The compiler was written in assembly language and used only 47 instructions types
- The Compiler Was Written By Very Smart IBM Programmers
- They Only Used 47 Out Of 140 Instructions IBM 360 Had For Writing The FORTRAN Compiler

- Only 47 Out of 140 Instructions Were Useful
- The Rest of The Instructions Added To The Complexity/Cost of IBM 360

Python: Basic Principles
The Name Comes from Monty Python

- Its core philosophy is summarized in the Zen of Python (PEP 20), which includes aphorisms such as:
  - Beautiful is better than ugly.
  - Explicit is better than implicit.
  - Simple is better than complex.
  - Complex is better than complicated.
  - Readability counts.
- Rather than building all of its functionality into its core, Python was designed to be highly extensible via modules. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications.
- Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Summary: Programming Language Evolution

- **FORTRAN (1957):** Developed by IBM, FORTRAN (Formula Translation) was the first high-level programming language designed for scientific and engineering calculations, making it easier to write numerical algorithms.
- **LISP (1958):** LISP (List Processing) was created for artificial intelligence research. It introduced the concept of symbolic expression processing and is still used in AI and symbolic computing.
- **COBOL (1959):** COBOL (COmmon Business-Oriented Language) was designed for business, finance, and administrative systems. It aimed to be readable and self-documenting.
- **ALGOL (1958-60):** ALGOL (ALGOrithmic Language) influenced many subsequent languages and introduced block structures, lexical scoping, and syntax formalism.
- **BASIC (1964):** Beginner's All-purpose Symbolic Instruction Code (BASIC) was developed for educational purposes and simplicity. It played a significant role in popularizing computer programming.
- **C (1972):** Developed at Bell Labs by Dennis Ritchie, C became a widely used language due to its efficiency and portability. It served as the foundation for the development of the UNIX operating system.
- **Pascal (1970):** Designed for teaching programming, Pascal introduced structured programming concepts and data structuring mechanisms.

Summary: Programming Language Evolution (continued)

- **C++ (1983):** An extension of C, C++ introduced object-oriented programming (OOP) features, allowing for better code organization and reuse.
- **Java (1995):** Developed by Sun Microsystems, Java was designed for cross-platform compatibility. It became popular for web development and enterprise applications.
- **Python (1991):** Known for its readability and simplicity, Python gained popularity for a wide range of applications, including web development, data science, and artificial intelligence..
- **JavaScript (1995):** Initially designed for web development, JavaScript evolved into a versatile language used for both client-side and server-side scripting.
- **Ruby (1995):** Known for its elegant syntax and focus on developer productivity, Ruby gained popularity, especially with the Ruby on Rails web framework.
- **C# (2000):** Developed by Microsoft, C# is part of the .NET framework and combines elements of C++ and Java. It's widely used for Windows application development
- **Rust (2010):** Known for its focus on memory safety without sacrificing performance, Rust is gaining popularity for systems programming.
- **Swift (2014):** Developed by Apple, Swift is designed for iOS, macOS, watchOS, and tvOS development. It aimed to be a more modern and safer alternative to Objective-C.
- **ATL (2022):** A programming language for high-performance computing

References (with direct web hypertext links)

- Assembly Language
- A Tensor Language to the Rescue | MIT CSAIL
- AA-KX10A-TC_RSTS_E_V9.5_PDP-11_MACRO-11_Language_Reference_Oct87.pdf (mirrorservice.org)
- Apollo Guidance Computer
- Compiler
- High Level Programming Language
- Interpreter
- Python Programming Language
- Zen of Python

B.W.Stuck Biography

- MIT, 1964-1972: SBEE 1968, SMEE 1969, EE 1970, PhD 1972
- Bell Laboratories 1972-1984:
  - Lectured at over 50 universities and research institutes in North America, Western Europe, Soviet Union, Japan.
  - Worked on 20 UNIX application systems, consulted on another 80, created a class for a masters level computer science program, which was taught at Columbia University three times, leading to publishing a book, A Computer and Communication Network Performance Analysis Primer, Prentice Hall, 1985
- Independent consulting, 1984-1998
- Venture capital, 1998-today