Today’s Topics

• Why We Need Programming Languages
  • Low-level Programming Language
  • High-level Programming Language

• How a Program Works
  • Compiler
  • Interpreters
Why We Need Programming Languages

• A computer’s CPU can only understand instructions that are written in machine language.
• Assembly language was created in the early days of computing as an alternative to machine languages.
• Instead of using binary numbers for instructions, assembly language uses short words that are known as mnemonics.
• Because assembly language is so close in nature to machine language, it is referred to as a low-level language.

```
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1
```
Why We Need Programming Languages

• People still find it very difficult to write entire programs in assembly language, other programming languages have been invented.
• Programming languages are notations for describing computations to people and to machines.
• In the 1950s, a new generation of programming languages known as high-level languages began to appear.
• They allow programmers to create powerful and complex programs without knowing how the CPU works, and without writing large numbers of low-level instructions.
# Some High-Level Programming Languages

<table>
<thead>
<tr>
<th>Languages</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC</td>
<td>Beginners All-purpose Symbolic Instruction Code, 1960s</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>FORmula TRANslator, 1950ss</td>
</tr>
<tr>
<td>COBOL</td>
<td>Common Business-Oriented Language, 1950s</td>
</tr>
<tr>
<td>Pascal</td>
<td>Originally designed for teaching programming, 1970s</td>
</tr>
<tr>
<td>C and C++</td>
<td>General purpose programming language, developed at Bell Lab in 1972 (C) and 1983 (C++)</td>
</tr>
<tr>
<td>C#</td>
<td>Around the year 2000 by Microsoft for .NET platform</td>
</tr>
<tr>
<td>Java</td>
<td>General purpose programming language, created by Sun Microsystem in early 1990s</td>
</tr>
<tr>
<td>JavaScript</td>
<td>Mainly used in web pages, created in 1990s</td>
</tr>
<tr>
<td>Pythons</td>
<td>General purpose programming language, created in the early 1990s</td>
</tr>
<tr>
<td>Ruby</td>
<td>General purpose programming language, created in the early 1990s</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>Created in the early 1990s for Windows-based applications</td>
</tr>
</tbody>
</table>
How a Program Works

• Before a program can be run, it first must be translated into a form in which it can be executed by a computer.
• The software systems that do this translation are called compilers.

• A compiler is a program that can read a program in one language – the source language – and translate it into an equivalent program in another language – the target language.
How a Program Works

- Compiler
  - Translates a source program into a target program
  - If the target program is an executable machine-language program, it can then be called by the user to process inputs and produce outputs
How a Program Works

• Phases of a compiler
  • Analysis part
    • breaks up the source program into constituent pieces
    • imposes a grammatical structure
    • uses this structure to create intermediate representation
  • Synthesis part
    • construct the desired target program from the intermediate representation and the information in the symbol table
  • Symbol table
    • the analysis part collects information about the source program and stores it in a data structure called a symbol table, which is passed along with the intermediate representation to the synthesis part.

Diagram:
- character stream
  - Lexical Analyzer
  - token stream
  - Syntax Analyzer
  - syntax tree
  - Semantic Analyzer
  - syntax tree
  - Intermediate Code Generator
  - intermediate representation
  - Machine-Independent Code Optimizer
  - Code Generator
  - intermediate representation
  - Machine-Dependent Code Optimizer
  - target-machine code
How a Program Works

position = initial + rate * 60

Symbol Table

<table>
<thead>
<tr>
<th></th>
<th>position</th>
<th>initial</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

character stream
Lexical Analyzer

token stream
Syntax Analyzer

syntax tree
Semantic Analyzer

syntax tree
Intermediate Code Generator

intermediate representation
Machine-Independent Code Optimizer

intermediate representation
Code Generator

target-machine code
Machine-Dependent Code Optimizer

target-machine code

character stream
Lexical Analyzer

Token stream
Syntax Analyzer

Syntax tree
Semantic Analyzer

Syntax tree
Intermediate Code Generator

Intermediate representation
Machine-Independent Code Optimizer

Intermediate representation
Code Generator

Target-machine code
Machine-Dependent Code Optimizer

Target-machine code

position = initial + rate * 60

<id, 1> => <id, 2> + <id, 3> * <60>

<id, 1> + <id, 2> * <id, 3> * 60

<id, 1> + <id, 2> * <id, 3> inttofloat 60

<id, 1> + <id, 2> * <id, 3> inttofloat 60
How a Program Works

position = initial + rate * 60

t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3

t1 = id3 * 60.0
id1 = id2 + t1

LDF  R2, id3
MULF R2, R2, #60.0
LDF  R1, id2
ADDF R1, R1, R2
STF  id1, R1

0001 0010 1111 0000
1110 0010 0011 1100
...
How a Program Works

- Interpreter
  - Translates a source program to its equivalent machine-language program and immediately executes them.

- The Python language uses an interpreter.
References & Photo Credits

• Pearson Custom Computer Science COMS W1001 Introduction to Information Science, Columbia University. Chapter 12 Introduction to Computer and Programming by Tony Gaddis

• Compilers, Principles, Techniques, and Tools. Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman.