Individual Exercises - Lecture 1

The following exercises you may prefer to do on your own, e.g. just after you have read the literature, and discuss the outcome with your group:

1. *Do Sebesta Review Questions 1 to 29 on page 54 and 55.*
    1. Why is it useful for a programmer to have the ability to learn new languages, even though he or she may have a good knowledge of a number of programming languages?

       Some languages are better suited for certain domains (Domain specific language)

       C would be more suited for operative systems, where python would be more suited for machine learning.

       Learning a language from a different paradigm (object oriented, functional, imperative) can give a deeper understanding of concepts that are available in most languages. For example learning a functional language like Haskell can improve your ability to utilize the functional programming features of Python.

    2. Why is it essential to choose an appropriate programming language for a specific software solution?

       The main advantage is increasing programmer productivity. As various languages are suited for different domains, a specific software solution should be written in a language, which suits that solution best.

       For example, writing safety critical applications for a bank, you might want to choose a language with a lot of compile time checks, to decrease risk of errors in the product.

    3. Which programming language for scientific applications was the first to be used successfully?

       Fortran

    4. Which is the first successful high-level programming language for business?

       COBOL

    5. In which programming language were most of the AI applications developed prior to 1990?

       LISP

    6. Which is the most popular markup language for Web development?

       HTML

7. Why is a list of programming language evaluation criteria for the development of software controversial?

   It is difficult to get people to agree on the ranking of criteria.

8. How does the overall simplicity of a programming language affect its readability?
   A language with a large number of basic constructs is more difficult to read/learn than one with a smaller number.
9. Why is the VAX instruction design orthogonal? By operating on registers and memory.

   VAX has only a small set of basic constructs, which then must work with each other in various ways to achieve more complex functionality.

10. Why does too much orthogonality cause problems?

    Too many possible combinations can lead to an increase in complexity.

11. Explain how “writability” is used as a measure of how easily a language can be used to create programs.

    For example a simple “Hello World!”-program in Java requires writing a class with public, static, void, main, i.e. learning both classes, access modifiers, static keywords, void return value and the main construct to simply print to the console. Printing to the console in java is also rather cumbersome requiring access to System.out.println(“Hello World!”);. Compare this to python, where one simply writes print(“Hello world!”).

12. Why is too much orthogonality a detriment to “writability”?
    Errors in programs can go undetected when nearly any combination of primitives is legal. E.g if operators have more uses in a single context, one could think an expression has one outcome, however the outcome might be another.
13. Give an example of expressivity in a language.
    ++i instead of i = i + 1 in the language C, C# and Java.
14. What is type checking?
    Testing for type errors in either compiling or run time. Some languages like Haskell include comprehensive type checking at compile time, which significantly decreases the risk of runtime errors. This, however, at the cost of writability.
15. Give an example of how the failure to type check, at either compile time or run time, can lead to countless program errors.

    If the typechecker fails to typecheck an assignment of incompatible types e.g. int i = “hello world”, a type is treated as a different type, which leads to countless errors.

16. How is the total cost of a programming language calculated?

    C

    The total cost of a programming language is the cost of learning, writing, compiling, executing and maintaining the code.

17. What is portability of a language?

    Portability is the ability to run programs on multiple different machines/architectures. Portability is most strongly influenced by the degree of standardization of the language. Some languages are not standardized at all, making programs in these languages very difficult to move from one implementation to another.

18. What is the use of the well-definedness criterion?

    The completeness and precision of the language's official defining document.

19. How does the execution of a machine code program on a von Neumann architecture computer occur?

    The execution of a machine code program on a von Neumann architecture computer occurs in a process called the fetch-execute cycle.

20. What two programming language deficiencies were discovered as a result of the research in software development in the 1970s?

    Incompleteness of type checking and inadequacy of control statements requiring the extensive use of gotos, which decreases readability and maintainability significantly.

21. What are the three fundamental features of an object-oriented programming language?

    Encapsulation, inheritance and polymorphism.

22. What language was the first to support the three fundamental features of object-oriented programming?

    Smalltalk

23. What is an example of two language design criteria that are in direct conflict with each other?

    Two criteria that conflict are reliability and cost of execution. For example, the Java language definition demands that all references to array elements be checked to ensure that the index or indices are in their legal ranges. This step adds a great deal to the cost of execution of Java programs that contain large numbers of references to array elements. C does not require index range checking, so C programs execute faster than semantically equivalent Java programs, although Java programs are more reliable. The designers of Java traded execution efficiency for reliability.

24. What are the three general methods of implementing a programming language?

    Compilation, pure-interpretation, hybrid-interpretation

25. Which produces faster program execution, a compiler or a pure interpreter?

    Compiler

26. What role does the symbol table play in a compiler?

    The symbol table serves as a database for the compilation process. The primary contents of the symbol table are the type and attribute information of each user-defined name in the program. This information is placed in the symbol table by the lexical and syntax analyzers and is used by the semantic analyzer and the code generator.

27. What does a linker do?

    The process of collecting system programs and linking them to user programs is called linking and loading, or sometimes just linking. It is accomplished by a systems program called a linker. In addition to systems programs, user programs must often be linked to previously compiled user programs that reside in libraries. So the linker not only links a given program to system programs, but also it may link it to other user programs.

28. Why is the von Neumann bottleneck important?

    The speed of the connection between a computer's memory and its processor often determines the speed of the computer, because instructions often can be executed faster than they can be moved to the processor for execution. This connection is called the von Neumann bottleneck

29. What are the advantages in implementing a language with a pure interpreter?

    More easily debugged, since the machine code can refer more directly to the actual code.

2. *Use Table 1.1 on page 31 in Sebesta's book to evaluate the C programming language*

   Simplicity: Few types, but several operators used for more tasks. i.e & and *

   Orthogonality: Low. For example arrays cannot be returned from a function, but can be parsed.

   Data types: Only primitives. Very few

   Syntax design: Originally, C was considered high level in its syntax. Since then, however, higher level languages has been developed, and C is now often no longer considered high level.

   Support for abstractions: Low. No classes or generics.

   Expressivity: low. Sorting, Strings, collections etc. require extensive programming.

   Type checking: Strict type checking at compile time.

   Exception handling: None, usually error codes in the form of ints are used

   Restricted aliasing: Yes, int, double etc. cannot be used as Identifiers.

3. *Use Table 1.1 on page 31 in Sebesta's book to evaluate the C# programming language*

Simplicity: C# has many constructs like Java, but then also constructs like structs and gotos. For this reason C# cannot be considered a simple language

Orthogonality: C# has relatively high orthogonality (The ability to combine primitive constructs of the language). For example 'x += y' is a shorthand for addition of the form x = x + y, but when used in the context of event publishers, it means adding a listener function 'y' to the event publisher 'x'.

Data types: Contains both primitive and non-primitive types like classes

Syntax design: Uses similar syntax to contemporary languages like C and Java in order to increase familiarity.

Support for abstraction: Yes, classes

Expressivity: Highly expressive (e.g compound statements, lists, strings)

Type checking: Strict type checking at compile time and runtime.

Exception handling: Yes (try, catch, finally)

Restricted aliasing: Yes, int, double etc. cannot be used as identifiers.

Table 1.1 Language evaluation criteria and the characteristics that affect them

	CRITERIA		
Characteristic	READABILITY	WRITABILITY	RELIABILITY
Simplicity	•	•	•
Orthogonality	•	•	•
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

4. *Do Fisher et al. exercise 3 on page 26 (exercise 5 page 55 in GE)*

C compilers are almost always written in C. This raises something of a "chicken and egg" problem—how was the first C compiler for a particular system created? If you need to create the first compiler for language X on system Y, one approach is to create a cross-compiler. A cross-compiler runs on system Z but generates code for system Y.

Explain how, starting with a compiler for language X that runs on systemZ, you might use cross-compilation to create a compiler for language X, written in X, that runs on system Y and generates code for system Y. What extra problems arise if system Y is "bare"—that is, has no

operating system or compilers for any language? (Recall that Unix R is written in C and thus must be compiled before its facilities can be used.)

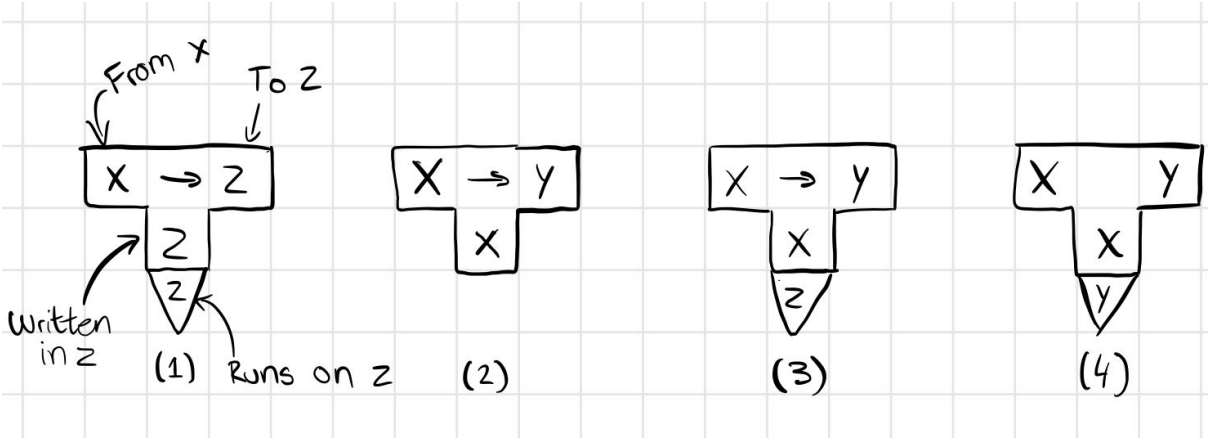
We are given a compiler(**1**) from X -> Z running on system Z.

To start off, create a compiler(**2**) from X -> Y written in X.

Next, compile the new compiler(**2**) using the given X -> Z compiler(**1**) that runs on Z.

Now you have a compiler(**3**) from X -> Y that is written in X, and runs on system Z.

Finally you can compile (**2**) using (**3**), which gives you a compiler(**4**) from X->Y written in X running on Y.



5. *Do Fisher et al. exercise 7 on page 28 (exercise 10 on page 57 in GE)*

Most C compilers (including the GCC compilers) allow a user to examine the machine instructions generated for a given source program. Run the following program through such a C compiler and examine the instructions generated for the for loop. Next, recompile the program, enabling optimization, and reexamine the instructions generated for the for loop. What improvements have been made? Assuming that the program spends all of its time in the for loop, estimate the speedup obtained. Write a suitable main C function that allocates and initializes a million-element array to pass to proc. Execute and time the unoptimized

and optimized versions of the program and evaluate the accuracy of your estimate.

```
int proc(int a[]) {
    int sum = 0, i;
    for (i=0; i < 1000000; i++)
       sum += a[i];
    return sum;
}
```

O0 optimization

```
push    rbp
mov     rbp, rsp
lea     rax, [rbp+16]
mov     QWORD PTR [rbp-8], rax
mov     eax, 0
pop     rbp
ret
```

O3 optimization

```
xor     eax, eax
ret
```

-O3 compiler optimization takes ~208 ms

-O0 compiler optimization takes ~ 250 ms

*Note that compilation time varies depending on the environment.

** O0 only optimizes compilation time and O3 has a range of optimization flags to be set, which includes executional optimization. Check the flags on https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

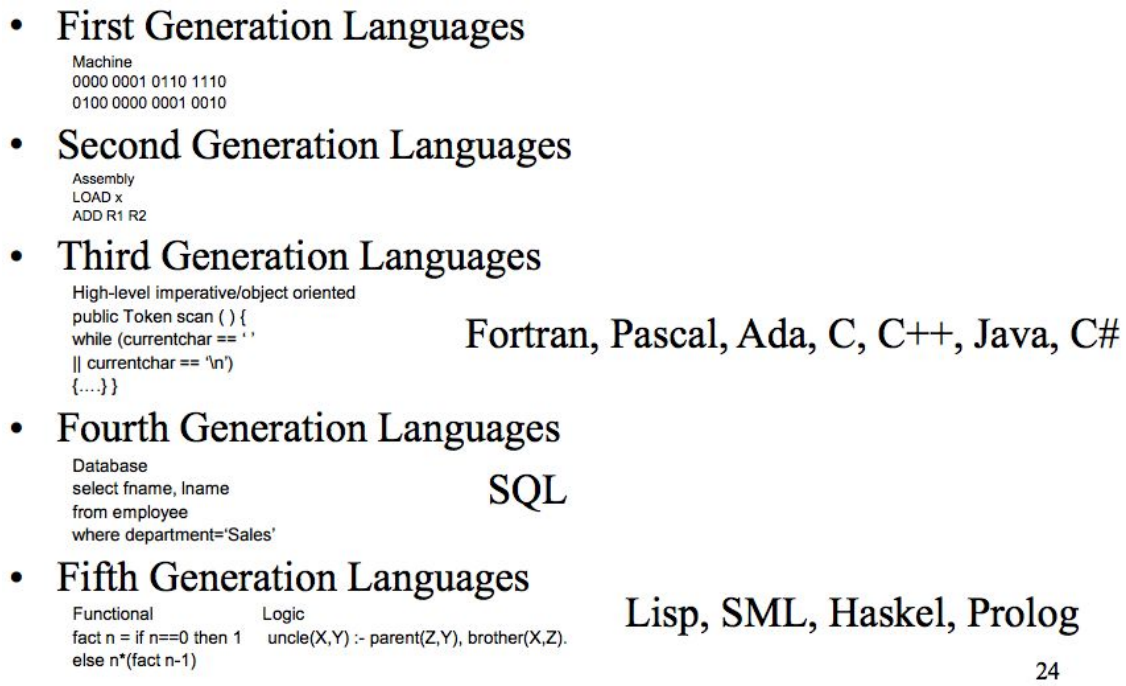
The compiler optimization -O3 speeds up the execution ~17%.

Group Exercises - Lecture 1

The following exercises are best done as group discussions:

1. *Review your individual list of programming languages and make a new list of all the computer languages that group members know.*
   Check your lists from lecture 0.

2. *Categorize the above languages according to generations (1st to 5th (or 6th))*



3. *Categorize the above languages according to paradigm (imperative, object oriented, declarative, special)*
   1. *Imperative: C, Fortran, Pascal, Ada (Also a little OOP)*
   2. *Object Oriented: C++, Java, C#*
   3. *Declarative: Haskel, SQL, Prolog, SML, LISP*

4. *Create a new list of language features group members would like in a new language. Are any of these features in conflict with each other? How would you prioritize the features?*
   Consider features like classes, typing, types, procedures, functions, lambda etc.
   Should it have a for loop? If so, how should it look?
   Example of a feature:
   ForForloop which executes like a nested for-loop
   for(int i = 0, int j = 0; i < 100, j < 3; i++, j++) { /* code */ }

5. *Do Sebesta exercise 1, 2, and 3 page 55.*
    1. *Do you believe our capacity for abstract thought is influenced by our language skills? Support your opinion.*

       Does an OOP language allow you to think in different abstractions than an imperative language?

    2. *What are some features of specific programming languages you know whose rationales are a mystery to you?*

       Maybe you don't see the rationale behind having both structs and classes, when classes can just as easily be used for the same purpose as structs?

    3. *What arguments can you make for the idea of a single language for all programming domains?*

       Having a single language for all domains would reduce fragmentation, but at the cost of excessive bloat. The language would be very big and complex, if it was to support all programming domains.

6. Do Sebesta exercise 4, 7, 10 and 17 page 56.

    4. *What arguments can you make against the idea of a single language for all programming domains?*

    Languages can be optimized for a specific domain, both in terms of performance, but also writability, readability, simplicity etc.

    7. *Java uses a right brace to mark the end of all compound statements. What are the arguments for and against this design?*

    Pros: Easier to see where a statement ends and multiple statements can be on a line.
    Cons: More code to write for the programmer without any added functionality.

    10. *What are the arguments for writing efficient programs even though hardware is relatively inexpensive?*

    Lower energy consumption and computation is never free. Additionally, it is sometimes difficult to assess if the hardware in the environment is indeed inexpensive and available with high performance.

    17. *Some programming languages—for example, Pascal—have used the semicolon to separate statements, while Java uses it to terminate statements. Which of these, in your opinion, is most natural and least likely to result in syntax errors? Support your answer.*

    Consider if semicolon would in other contexts be considered a separator of elements (or is it comma?).

7. *Do Sebesta exercise 18 page 56.*

   18. Many contemporary languages allow two kinds of comments: one in which delimiters are used on both ends (multiple-line comments), and one in which a delimiter marks only the beginning of the comment (one-line comments). Discuss the advantages and disadvantages of each of these with respect to our criteria.

   One-line: less typing for single line comments but more work to comment out multiple lines

   Multiple-line: less typing to comment out multiple lines but more typing for one. Multiline comments can also serve to increase writability, since it allows to quickly comment out sections of code for debugging purposes.

8. *Do Fisher et al. Exercise 5 and 6 on page 27-28 (exercise 7 and 14 on page 56 resp. 58 in GE)*

   5. To allow the creation of camera-ready documents, languages like TeX and LaTeX have been created. These languages can be thought of as varieties of programming languages whose output controls a printer or display. Source language commands control details like spacing, font choice, point size, and special symbols. Using the syntax-directed compiler structure of Figure 1.4, suggest the kind of processing that might occur in each compiler phase if TeX or LaTeX input was being translated.

   An alternative to “programming” documents is to use a sophisticated editor such as that provided in Microsoft Word or Adobe FrameMaker to interactively enter and edit the document. (Editing operations allow the choice of fonts, selection of point size, inclusion of special symbols, and so on.) This approach to document preparation is called **WYSI-WYG**—what you see is what you get—because the exact form of the document is always visible.

   What are the relative advantages and disadvantages of the two ap- proaches? Do analogues exist for ordinary programming languages?

   WYSIWYG editors provide an easy interface that most people can use, while LaTeX makes it easier to automate creation of documents so they can be made from templates (Class files).

6. Although compilers are designed to translate a particular language, they often allow calls to subprograms that are coded in some other language (typically, Fortran, C, or assembler). Why are such “foreign calls” allowed? In what ways do they complicate compilation?

FFI allows a programming language to interact with another language, which makes it possible to use foreign libraries and services. This can be useful if the “host” programming language is too slow (like Python) or if it does not provide this functionality at all. Compilation is complicated, since now the compiler has to interleave the output of the two languages to create the result.

9. *Do Fisher et al. Exercise 8 on page 28 (exercise 11 on page 57 in GE)*

8. C is sometimes called the ***universal assembly language*** *in light of its ability to be very efficiently implemented on a wide variety of computer architectures. In light of this characterization, some compiler writers have chosen to generate C code as their output instead of a particular machine language. What are the advantages to this approach to compilation? Are there any disadvantages?*

Advantages: Easy portability. C can be compiled to fast assembly code with existing C compilers. Now the compiler writes do not have to replicate all of the optimizations already built into the C compiler.

Disadvantages: It might be difficult to translate some high-level abstractions to C. You lose low level control, and you might not be able to fully optimize the performance of the new features that are in your language.