Queens College of CUNY
Department of Computer Science
Programming Languages
(CSCI 316)
Winter 2026

Assignment #7
"Midterm Exam Review Sheet"
Due: January 12, 2026

Introduction and Instructions:

In this assignment, we use "crowd-sourcing" (i.e. class participation and partnership) to populate and prepare this review sheet for tomorrow's Midterm Exam.

Using your own knowledge already obtained; referring back to the book, slides, and lecture notes;or using online tools such as internet search and AI chats, complete as many (unanswered) review questions as you can *in this document*. Next to each answer you provide, place your full name in parentheses so you can be credited for your contributions.

To review different areas, it is recommended that you answer at least one question from each chapter. You may submit an answer to a question already answered - based on independent research or resource - only if all review questions from that chapter have already been answered at least once each.

Submissions

You do not need to submit anything in the Google form for this assignment. The assignment will be scored on the basis of the quantity and quality of the review questions you have answered in this document during the in-class activity.

Chapter 1: Preliminaries

Summary

The study of programming languages is valuable for some important reasons:

- increases our capacity to use different constructs in writing programs,
- enables us to choose languages for projects more intelligently
- makes learning new languages easier.

Computers are used in a wide variety of problem-solving domains. The design and evaluation of a particular programming language is highly - dependent on the domain in which it is to be used.

Among the most important criteria for evaluating languages are readability, writability, reliability, and overall cost. These will be the basis on which we examine and judge the various language features discussed in the remainder of the book.

The major influences on language design have been machine architecture and software design methodologies.

Designing a programming language is primarily an engineering feat, in which a long list of trade-offs must be made among features, constructs, and capabilities.

The major methods of implementing programming languages are compilation, pure interpretation, and hybrid implementation.

Programming environments have become important parts of software development systems, in which the language is just one of the components.

Review Questions

1. Why is it useful for a programmer to have some background in language design, even though he or she may never actually design a programming language?

Most languages are built from common language ideas. If you understand those ideas, learning new languages will become much easier. -Kaiwen Liu

2. How can knowledge of programming language characteristics benefit the whole computing community?
Leads to better language design, optimal tool selection, and reduces learning curve of new languages and technologies. (Jigme Topgyal)

3. What programming language has dominated scientific computing over the past 60 years?
FORTRAN (SONU KHADGI)
Fortran, historically Emphasized Floating point arithmetic and arrays (Nicholas Roberts)

4. What programming language has dominated business applications over the past 60 years?
COBOL (Vincent Comaianni)

5. What programming language has dominated artificial intelligence over the past 60 years?
LISP (SONU KHADGI)

6. In what language is most of UNIX written?
C (SONU KHADGI)
Most of UNIX is written in C (Lily Zheng)
Thanks Dennis Ritchie (Jeremy Haft.)

7. What is the disadvantage of having too many features in a language?
REDUCES READABILITY, HARD TO BUILD AND MAINTAIN WHILE COMPILING (SONU KHADGI)
While not as direct as readability, writability is also reduced as the learning curve would be higher having to learn a lot more in the language. (Vegus Tao)

8. How can user-defined operator overloading harm the readability of a program?
Three key reasons as to why user-defined operator overloading can harm the readability of a program:
Overloading an operator to contradict its inherent or assumed meaning, an example of this is overloading the multiplication operator to function as concatenation for strings
It can cause simple operations to have efficiency problems, for example causing an operator such as ‘+’ to create an entire complex memory allocation, or database query, when the intent is simply to add
Safety reasons, bad coding of an overload for an operator can cause problems with mathematical properties causing headaches for readers of said code, for example: object1 + object2 would work, but object2 + object1 may not as intended.
(Vincent Comaianni)

9. What is one example of a lack of orthogonality in the design of C?

C has two kinds of structured data types, arrays and structs(records), structs can be returned from functions but arrays cannot. (Kristopher Garcia)

10. What language used orthogonality as a primary design criterion?
A language that used orthogonality as a primary design criterion is ALGOL 68. (Kaiwen Liu)

11. What primitive control statement is used to build more complicated control statements in languages that lack them?
The primitive control statement that was used to build more complicated control statements in languages that lack them is GOTO statement. GOTO was the building block before structured control statements existed.
(Jeaneth Lliguicota)

12. What does it mean for a program to be reliable?
A program is reliable when it consistently behaves correctly and predictably under expected conditions and handles unexpected situations without failing. (Arsalan Ansari).

13. Why is type checking the parameters of a subprogram important?
Type checking the parameters of a subprogram is important because it ensures that arguments passed to the subprogram are compatible with the expected types, preventing errors, improving program reliability, and allowing many mistakes to be detected at compile time. (Jahid Hasan)

14. What is aliasing?
TWO OR MORE VARIABLES REFER TO A SAME MEMORY LOCATION MAKING PROGRAM HARD TO PREDICT (SONU KHADGI)
Multiple names pointing to the same memory location. (Jackie Lin)

15. What is exception handling?
Exception Handling is used to detect, manage, and respond to runtime errors so that the program doesn't randomly crash. It's important because it improves reliability and makes maintenance easier. (Oscar Yang)

16. Why is readability important to writability?
You can't easily write what you can't easily read. When language constructs are clear and predictable, programmers can compose correct code faster and with fewer mistakes. (Jahid Hasan)

17. How is the cost of compilers for a given language related to the design of that language?

Languages with complex syntax, many special cases, and irregular features require more sophisticated parsing and semantic analysis, which increases development and maintenance costs. In contrast, languages that emphasize simplicity, consistency, and orthogonality are easier to specify and implement, resulting in lower compiler costs. (Jahid Hasan)

19. What is the name of the category of programming languages whose structure is dictated by the von Neumann computer architecture?
Imperative programming language (Jigme Topgyal)

20. What two programming language deficiencies were discovered as a result of the research in software development in the 1970s?
Lack of Abstraction and Modularity and Lack of Reliability and Safety (Adeel Asaf)

21. What are the three fundamental features of an object-oriented programming language?
Encapsulation, inheritance, polymorphism (Lily Zheng)

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

Smalltalk (Lily Zheng)

23. What is an example of two language design criteria that are in direct conflict with each other?
Writeability and reliability. E.g Pointers allow data addressing directly but are dangerous for reliability and even readability. (Kristopher Garcia)

24. What are the three general methods of implementing a programming language?
Compilation- translate the whole program into machine code before running it.
Interpretation- execute the program directly by reading and running it step-by-step
Hybrid/VM
(Azmain A)

25. Which produces faster program execution, a compiler or a pure interpreter?
Answer: **compiler** produces faster program execution
Turns source code into machine code before running
Optimize code during translation
Direct execution w/o needing another program interpretation (Monika Luchowska)

26. What role does the symbol table play in a compiler?
The symbol table is built during the compilation process and is used for tasks like type checking, scope resolution, and code optimization. (Jude Pierre)

27. What does a linker do?
A linker is a software tool that combines varied pieces of code and libraries into a single runnable program (Andrew Laboy)

28. Why is the von Neumann bottleneck important?
The connection speed between a computer's memory and its processor determine the speed of a computer, however program instruction can often be executed much faster than the speed of the connection, which is a limiting factor in the speed of computers. (Jackie Lin)

29. What are the advantages in implementing a language with a pure interpreter?
Some advantages in implementing a language with a pure interpreter would be it not needing a translation into machine code and its easier to implement programs. (Jackie Lin)

Chapter 2: Evolution of the Major Programming Languages

Summary

We have investigated the development of a number of programming languages. This chapter gives the reader a good perspective on current issues in language design. We have set the stage for an in-depth discussion of the important features of contemporary languages.

Conceptual Overview

The evolution of programming languages reflects the changing needs of computing—from early scientific calculation to business data processing, abstraction, object orientation, scripting, and web development. Each major language emerged to solve particular problems, and in doing so, introduced concepts that shaped later designs.

Early Foundations: Machine Independence and High-Level Thought

The earliest computers were programmed directly in machine or assembly language, tightly coupled to hardware and extremely error-prone. The first major conceptual leap was Konrad Zuse’s Plankalkül (1940s), the earliest high-level language design. Although never implemented at the time, it introduced forward-looking ideas such as variables, structured data, and expressions, demonstrating that programming could be abstracted from hardware.

As computing matured, pseudocode emerged as an informal way to express algorithms independently of any specific machine. While not executable, pseudocode influenced the mindset of language designers by emphasizing clarity and algorithmic structure over hardware details.

FORTRAN and the Scientific Computing Era

The first widely adopted high-level language was FORTRAN (1957), developed for the IBM 704. Designed for scientific and engineering computation, FORTRAN emphasized efficiency, numeric computation, and array processing. Its success proved that high-level languages could rival assembly language in performance. FORTRAN introduced compiled languages, looping constructs, and formula-style expressions, setting a precedent for future scientific languages.

Lisp and Functional Programming

In contrast to FORTRAN’s numeric focus, Lisp (1959) was designed for symbolic computation and artificial intelligence research. Lisp pioneered functional programming, recursion, dynamic typing, automatic memory management (garbage collection), and list processing. Its core ideas—code as data, higher-order functions, and recursion—became foundational not only for functional languages but also for modern scripting and multi-paradigm languages.

ALGOL and Structured Programming

ALGOL 60 represented a major advance in language design philosophy. It introduced block structure, lexical scoping, nested subprograms, and a formal syntax description using Backus–Naur Form (BNF). Although ALGOL itself never achieved commercial dominance, it deeply influenced language theory and design, shaping descendants such as Pascal, Ada, and C. ALGOL emphasized readability, mathematical clarity, and structured programming.

Business Computing: COBOL

While FORTRAN and ALGOL targeted scientific users, COBOL (1960) was created for business data processing. Its defining feature was English-like syntax, designed to make programs readable by non-specialists. COBOL introduced record structures, decimal arithmetic, and file processing, making it ideal for payroll, accounting, and large-scale data management. Despite criticism for verbosity, COBOL’s longevity underscores the importance of domain-specific language design.

Timesharing and Beginner Languages: BASIC

With the rise of timesharing systems, BASIC was developed to provide easy access to computing for students and non-experts. BASIC prioritized simplicity and interactivity, often at the expense of structure. While early versions encouraged poor programming practices (such as unrestricted GOTO), BASIC played a crucial role in democratizing programming and later evolved into more structured dialects.

Multipurpose and Experimental Languages

PL/I attempted to unify scientific and business programming in a single language, offering extensive features for both domains. However, its complexity limited widespread adoption. Around the same time, dynamic and experimental languages such as APL and SNOBOL explored powerful operators and string processing, demonstrating high expressiveness but also highlighting trade-offs between power and readability.

Data Abstraction and Object Orientation

The introduction of SIMULA 67 marked the birth of object-oriented programming, introducing classes, objects, inheritance, and dynamic binding. These concepts shifted programming from procedures to modeling real-world entities.

ALGOL 68 extended ALGOL's principles with extreme orthogonality and strong typing. Although theoretically elegant, its complexity limited practical use, illustrating the danger of excessive generality in language design.

Logic Programming: Prolog

Prolog emerged from research in automated theorem proving and logic programming. Programs are written as facts and rules, with execution driven by logical inference rather than explicit control flow. Prolog demonstrated that programming could be declarative, influencing later constraint-based and rule-based systems.

Large-Scale Design and Reliability: Ada

Developed under U.S. Department of Defense sponsorship, Ada was one of the most ambitious language design efforts ever undertaken. It emphasized strong typing, modularity, concurrency, and reliability, making it suitable for large, safety-critical systems. Ada highlighted how language design could directly address software engineering concerns such as maintainability and correctness.

Object-Oriented Mainstream: Smalltalk, C++, and Java

Smalltalk refined object-oriented concepts into a pure object model, influencing both academic research and industrial practice

Review Questions

1. In what year was Plankalkül designed? In what year was that design published?
Plankalkul was designed in 1945, and was published in 1972. (Jackie Lin)

2. What two common data structures were included in Plankalkül?
The two common data structures used in Plankalkul are Arrays and Records. (Arsalan Ansari).

3. How were the pseudocodes of the early 1950s implemented?
Through interpretation (specifically, software interpreters)（Junbao Zhang)

4. Speedcoding was invented to overcome two significant shortcomings of the computer hardware of the early 1950s. What were they?
The lack of floating-point hardware and the difficulty of manual address incrementing.(Junbao Zhang)

5. Why was the slowness of interpretation of programs acceptable in the early 1950s?

- Because in the early 1950s computer time was dominated by human/setup time, not execution speed. (Azmain A)

6. What hardware capability that first appeared in the IBM 704 computer strongly affected the evolution of programming languages? Explain why.
    The hardware capability that first appeared in the IBM 704 computer strongly affected the evolution of programming languages is index registers. Index register made it easy to access arrays elements and modify addresses automatically in loops. Lead to support for arrays, loop constructs, and efficient complied code. (Jeaneth Lliguicota)

7. In what year was the Fortran design project begun?
1954 was the year when the Fortran design project begun (Adeel Asaf)

8. What was the primary application area of computers at the time Fortran was designed?
Scientific and Computation.(Kaiwen Liu)

9. What was the source of all of the control flow statements of Fortran I?
Mathematic (Sonu Khadgi)

10. What was the most significant feature added to Fortran I to get Fortran II?
Fortran II added independent compilations. (Jackie Lin)

11. What control flow statements were added to Fortran IV to get Fortran 77?
The If, then, else statement (Vegus Tao)

12. Which version of Fortran was the first to have any sort of dynamic variables?
Fortran 90 (Gyeonghwan Do)

13. Which version of Fortran was the first to have character string handling?
Fortran 77 (Gyeonghwan Do)

14. Why were linguists interested in artificial intelligence in the late 1950s?
Linguists were concerned with natural language processing (Andrew Laboy)

15. Where was Lisp developed? By whom?

- Lisp was developed at MIT by John McCarthy in 1958. (Oscar Yang)

16. In what way are Scheme and Common Lisp opposites of each other?
Scheme prioritizes minimalist elegance, conceptual purity, and a single namespace (Lisp-1), enforcing lexical scoping and tail-call optimization for functional programming, while Common Lisp is a large, complex, multi-paradigm language with separate namespaces for functions and variables (Lisp-2), allowing dynamic scoping and offering extensive features for imperative programming alongside functional ones, essentially embodying opposite ends of the Lisp spectrum from minimalist to maximalist, pure functional to multi-paradigm. (Weitong Chen)

17. What dialect of Lisp is used for introductory programming courses at some universities?
Scheme - (Dariel Urena)

18. What two professional organizations together designed ALGOL 60?
Answer:**ACM** Association for Computing Machinery
GAMM — *Gesellschaft für Angewandte* (Shiwlee Rahman)

19. In what version of ALGOL did block structure appear?
Appeared in ALGOL 60 (Vincent Comaianni)
'
20. What missing language element of ALGOL 60 damaged its chances for widespread use?
Answer: was **standardized input and output (I/O) statements. (Shiwlee Rahman)**

21. What language was designed to describe the syntax of ALGOL 60?
BNF (Backus-Naur Form) - (Dariel Urena)

22. On what programming language was COBOL based?
FLOW-MATIC (Weitong Chen)

23. In what year did the COBOL design process begin?
1959 (Weitong Chen)

24. What data structure that appeared in COBOL originated with Plankalkül?
Hierarchical data structures or records (Weitong Chen)

25. What organization was most responsible for the early success of COBOL (in terms of extent of use)?
U.S. Department of Defense, through the CODASYL (Conference on Data Systems Languages) (Weitong Chen)

26. What user group was the target of the first version of Basic?
The first target group of Basic was for non-technical users to be able to have access to computers (Joseph Pasqualicchio)

27. Why was Basic an important language in the early 1980s?
Basic was one of the first programming languages that became accessible to everyday people (Joseph Pasqualicchio)

28. PL/I was designed to replace what two languages?
Fortran and COBOL - (Dariel Urena)

29. For what new line of computers was PL/I designed?
Shiwlee Rahman
Answer: PL/I was designed for IBM's new System/360 line of computers

30. What features of SIMULA 67 are now important parts of some object-oriented languages?
Creating classes and objects. (Vegus Tao)

31. What innovation of data structuring was introduced in ALGOL 68 but is often credited to Pascal?

a) ALGOL 68 created complex data structures
    i) **By combining primitive types**
b) Pascal simplified by making "records"
    i) Define custom data types
    ii) Ex: "Person" w/ name + age
    iii) Hint: called "structs" in C (Monika Luchowska)

32. What design criterion was used extensively in ALGOL 68?
Answer: The design criterion used extensively in ALGOL 68 was orthogonality. (Shiwlee Rahman)

33. What language introduced the case statement?
The language which introduced the case statement was ALGOL 60, in the form of begin and end blocks.
(Vincent Comaianni)

34. What operators in C were modeled on similar operators in ALGOL 68?
Compound assignments; where you can conduct an operation and assign the value to a variable on the same line (+=, -=, *=, /=). (Vegus Tao)

35. What are two characteristics of C that make it less safe than Pascal?
C allows **direct pointer arithmetic and unrestricted memory access**, whereas Pascal restricts pointer use to ensure memory safety.
C does **NOT perform automatic array bounds checking**, which can lead to buffer overflows and crashes that Pascal's strict runtime checks prevent. (Cheuk Yiu Sung)

36. What is a nonprocedural language?
A language that specifies what to produce, not how to produce. (Sonu Khadgi)

37. What are the two kinds of statements that populate a Prolog database?
Two kinds of statements that populate a Prolog database are facts and rules. Facts represent unconditional truths, and state relationships or properties. Rules represent conditional relationships, and define facts in terms of other facts. (Oscar Yang)

38. What is the primary application area for which Ada was designed?
Large scale design and reliability, developed by US DoD sponsorship. (Kristopher Garcia)

39. What are the concurrent program units of Ada called?
Tasks. -Kristopher Garcia

40. What Ada construct provides support for abstract data types?
Packages. An abstract data type is implemented as a package that includes the type and its operations (Nicholas Roberts)

41. What populates the Smalltalk world?
Objects since it is a pure object language - (Dariel Urena)

42. What three concepts are the basis for object-oriented programming?
Polymorphism, Encapsulation, Inheritance(Jeremy Haft)

43. Why does C++ include the features of C that are known to be unsafe?
Pointers are very handy for direct addressing(Jeremy Haft)
'
44. What language was Swift designed to replace?
Objective - C (Jeremy Haft)

45. What do the Ada and COBOL languages have in common?
They were both high level languages that were critical early on and which the DoD had a hand in developing. Ada is an object oriented language that emphasized on reliability and safety while COBOL focused on business. - (Dariel Urena)

46. What was the first application for Java?
A TV Remote (Jeremy Haft)

47. What characteristic of Java is most evident in JavaScript?
Objects? JSON very object focused. (Jeremy Haft)

48. How does the typing system of PHP and JavaScript differ from that of Java?
Answer:

a) **PHP and JavaScript use dynamic typing**
    i) Variables are associated w/ values
        1) Not data types
    ii) Implicit conversion
        1) Automatic type conversion
    iii) Type checking at runtime
        1) Operations validity checked when code is executed
b) **Java uses static typing**
    i) Only hold values of specific data type
    ii) Type checking at compile time
    iii) Fewer runtime type errors
    iv) Explicit conversion
        1) Done manually / cast by programmer (Monika Luchowska)

49. What array structure is included in C# but not in C, C++, or Java?
A true multidimensional rectangular array. (Jigme Topgyal)

50. What two languages was the original version of Perl meant to replace?
awk and sh. (Jigme Topgyal)

51. For what application area is JavaScript most widely used?
JavaScript is used for everything these days in different formats but for a long time it started out and was mainly used as a scripting language to be used for interactive web applications / websites (Joseph Pasqualicchio)

52. What is the relationship between JavaScript and PHP, in terms of their use?
JavaScript is mainly used for client-side scripting and PHP is used for server-side scripting. They can be used together to handle both the frontend and backend for a website. (Joseph Pasqualicchio)

53. PHP's primary data structure is a combination of what two data structures from other languages?
PHPs primary data structure is a combination of both indexed and associative arrays, or the generic lists and hashmaps into one data structure. (Joseph Pasqualicchio)
An example can look something like:
```
$data = array(
 0 => “zero”, // (indexed)
 “color” => “red” // (associative)
)
```

54. What data structure does Python use in place of arrays?
List (Lily Zheng)

55. What characteristic does Ruby share with Smalltalk?
Answer: **both are object-oriented language**

a) Objects include: numbers, classes, control structure
b) Message passing via method calls to objects
c) Dynamic typing - types checked at runtime (Monika Luchowska)

56. What characteristic of Ruby's arithmetic operators makes them unique among those of other languages?
The characteristic of Ruby's arithmetic operators which makes them unique among those of other languages is that they're in actuality methods in Ruby, not operators. (Vincent Comaianni)

57. What deficiency of the switch statement of C is addressed with the changes made by C# to that statement?
C# improves reliability by prohibiting implicit fall-through, requiring every non-empty case block to end with an explicit jump statement like break to prevent accidental logic errors. (Cheuk Yiu Sung)

58. What is the primary platform on which C# is used?
.NET (Vegus Tao)

59. What are the inputs to an XSLT processor?
An XML document (source document) and an XSLT stylesheet (Adeel Asaf)

60. What is the output of an XSLT processor?
A transformed document, usually HTML, XML, or plain txt (Adeel Asaf)

61. What element of the JSTL is related to a subprogram?

The element of the JSTL that related to a subprogram is custom tag elements. A custom tag in JSTL encapsulates behavior, can be reused, is invoked with parameters, and performs a specific task.

(Jeaneth Lliguicota)

62. To what is a JSP document converted by a JSP processor?

- A JSP (JavaServer Pages) document is converted by a JSP processor into a Servlet. (Oscar Yang)

63. Where are servlets executed?
On a web server. (Sonu Khadgi)

Chapter 3: Describing Syntax and Semantics

Summary

Backus-Naur Form and context-free grammars are equivalent metalanguages that are well suited for the task of describing the syntax of programming languages. Not only are they concise descriptive tools, but also the parse trees that can be associated with their generative actions give graphical evidence of the underlying syntactic structures. Furthermore, they are naturally related to recognition devices for the languages they generate, which leads to the relatively easy construction of syntax analyzers for compilers for these languages.
An attribute grammar is a descriptive formalism that can describe both the syntax and static semantics of a language. Attribute grammars are extensions to context-free grammars. An attribute grammar consists of a grammar, a set of attributes, a set of attribute computation functions, and a set of predicates that describe static semantics rules.
This chapter provides brief introductions to three methods of semantic description: operational, denotational, and axiomatic.

- Operational semantics is a method of describing the meaning of language constructs in terms of their effects on an ideal machine.
- In denotational semantics, mathematical objects are used to represent the meanings of language constructs. Language entities are converted to these mathematical objects with recursive functions.
- Axiomatic semantics, which is based on formal logic, was devised as a tool for proving the correctness of programs.

Review Questions

1. Define syntax and semantics.
Syntax is the study of grammar or the structure of a language, independent of the meaning of the language entities. **Semantics** is the study of meaning. For the same semantic concept, different languages will employ different syntaxes to construct sentences. E.g.:x+=y(C++.etc) OR ADD x,y (asm language) To use an example from natural language: "*The dog is a man.*" From a **syntactic** perspective, this sentence is *correct*; it contains a subject, a verb, and an object, and the grammatical components (such as gender or case) are handled properly. However, from a **semantic** perspective, it is incorrect because *a dog cannot be a man*. (Cheuk Yiu Sung)

2. Who are language descriptions for?
Language descriptions are for the people that need to use the language, to understand, use, and implement a programming language. (Arsalan Ansari).

3. Describe the operation of a general language generator.
A general language generator operates through defining a set of rules which define its grammar, and which are operated on by a general language recognizer. (Vincent Comaianni)

4. Describe the operation of a general language recognizer.
A general language recognizer takes an input string and checks it against the rules of a language. (Adeel Asaf)

5. What is the difference between a sentence and a sentential form?

A sentential form may contain nonterminals, whereas a sentence only contains terminals and is a complete string in that language.(Kaiwen Liu)

6. Define a left-recursive grammar rule.
A production where a non-terminal symbol immediately or indirectly refers to itself as the first symbol on the right-hand side, creating a loop like A → Aα | β, which defines a repetitive structure (e.g., expressions like expr + term) -Kristopher Garcia

7. What three extensions are common to most EBNFs?
EBNFS extensions are: optional parts are placed in [ ], alternative parts are in ( ), and zero or more repetitions are in { }. (Jackie Lin)

8. Distinguish between static and dynamic semantics.
Static semantics define the compile-time rules of a programming language, such as type checking, scope rules, and declaration requirements, which help ensure program correctness before execution. Dynamic semantics describe the run-time behavior of programs, explaining how statements are executed and how program state changes during execution. (Jahid Hasan)

9. What purpose do predicates serve in an attribute grammar?
In an attribute grammar, predicates are used to check conditions that must be true for a program to be correct. They help enforce rules like making sure variables are used with the right types or that certain values are allowed in specific situations. (Jahid Hasan)

10. What is the difference between a synthesized and an inherited attribute?
Synthesized attributes get their values from children nodes while inherited attributes get their values from the parent or sibling nodes. (Vegus Tao)

11. How is the order of evaluation of attributes determined for the trees of a given attribute grammar?
Identify Attributes and Rules -> Build a dependency graph -> Apply Topological Sort
(Jeremy Haft)

12. What is the primary use of attribute grammars?
The primary use of attribute grammars is to define the semantics of programming languages. It’s used to specify semantic rules associated with the syntax. They extend CFGs by attaching meaning to syntax. (Oscar Yang)

13. Explain the primary uses of a methodology and notation for describing the semantics of programming languages.
Methodology and notation is used to give precise and unambiguous meanings to programs that everyone can interpret the language in the same way. (Tony Chen)

14. Why can machine languages not be used to define statements in operational semantics?
The individual steps in the execution of machine language and the resulting changes to the state of the machine are too small and too numerous. Second the storage of a real computer into large and complex. (Andrew Laboy)

15. Describe the two levels of uses of operational semantics.
Operational semantics describes the meaning of a program by specifying how said program executes. It can be broken down as well into two levels of uses of operational semantics, which are Big-Step Semantics and Small-Step Semantics.

Big-Step Semantics: focuses on the final result of computation, it understands the execution of a loop or function to be simply one single but large step, unconcerned with how the final result was reached or the steps taken inside of the loop or function.

Small-Step Semantics: focuses on the individual small statements and their workings, it describes how a simple statement such as “x + 2” changes machine registers or memory.

(Vincent Comaianni)

16. In denotational semantics, what are the syntactic and semantic domains?
Domain is the collection of values that are legitimate parameters to the function. The range is the collection of objects to which the parameters are mapped. In denotational semantics the domain is called syntactic domain because it is syntactic structures that are mapped. The range is called the semantic domain. (Andrew Laboy)

The syntactic domains are the input grammar CFG. The semantic domain is the mathematical relationships that give meaning to the syntactical pieces.(Jeremy Haft)
17. What is stored in the state of a program for denotational semantics?
Answer: **current value of all its variables**

a) Just like in class assignment, environment map forms
    i) Var names (IDs)
    ii) Corresponding values (Monika Luchowska)

18. Which semantics approach is most widely known?
The most widely known semantics approach is operational semantics. Operational semantics explains what a program does step by step. (Jeaneth Lliguicota)

19. What two things must be defined for each language entity in order to construct a denotational description of the language?
Syntax form and Semantic mapping function. (Sonu Khadgi)

20. Which part of an inference rule is the antecedent?
Answer: In an inference rule, the antecedent is the condition (or premise) part—the part that appears before the arrow (→) or above the line in rule notation. ((Shiwlee Rahman)

21. What is a predicate transformer function?
A function that maps post-conditions to pre-conditions. (Sonu Khadgi)

22. What does partial correctness mean for a loop construct?
Partial correctness in a loop structure means that the result that was produced has met the required condition. Therefore the loop terminates (Lily Zheng)

23. On what branch of mathematics is axiomatic semantics based?
Axiomatic semantics are based on predicate logic (Nicholas Roberts)

24. On what branch of mathematics is denotational semantics based?
Recursive function theory. (Jigme Topgyal)

25. What is the problem with using a software pure interpreter for operational semantics?
The problem with using a software pure interpreter for operational semantics is that it makes the program much slower having to interpret the code for each operation compared to having a fully compiled language with things handled before runtime. (Joseph Pasqualicchio)

26. Explain what the preconditions and postconditions of a given statement mean in axiomatic semantics.
The precondition is what must be true before execution, the weakest precondition is the least restrictive precondition that makes the postcondition true, and postcondition is what is true after execution. (Jackie Lin)

27. Describe the approach of using axiomatic semantics to prove the correctness of a given program.
Answer: axiomatic semantics are **approach to proving correctness of programs**

a) Specifying commands' properties
b) Logical assertions
    i) Describe state of program
        1) Before + after execution
c) Defining
    i) Preconditions
    ii) Postconditions (Monika Luchowska)

28. Describe the basic concept of denotational semantics.
Denotational semantic describes the meaning of a program by mapping each language construct to a mathematical object that represent what it does, rather than how it executes step by step. (Adeel Asaf)

29. In what fundamental way do operational semantics and denotational semantics differ?
Operational is more of explaining the program by describing how it executes step by step on an abstract machine, while denotational semantics explains a program by mapping it to a mathematical function. (Adeel Asaf)

Chapter 4: Lexical and Syntax Analysis

Summary

Syntax analysis is a common part of language implementation, regardless of the implementation approach used. Syntax analysis is normally based on a formal syntax description of the language being implemented. A context-free grammar, which is also called BNF, is the most common approach for describing syntax. The task of syntax analysis is usually divided into two parts: lexical analysis and syntax analysis. There are several reasons for separating lexical analysis—namely, simplicity, efficiency, and portability.

A lexical analyzer is a pattern matcher that isolates the small-scale parts of a program, which are called lexemes. Lexemes occur in categories, such as integer literals and names. These categories are called tokens. Each token is assigned a numeric code, which along with the lexeme is what the lexical
analyzer produces.

There are three distinct approaches to constructing a lexical analyzer: using a software tool to generate a table for a table-driven analyzer, building such a table by hand, and writing code to implement a state diagram description of the tokens of the language being implemented. The state diagram for tokens can be reasonably small if character classes are used for transitions, rather than having transitions for every possible character from every state node. Also, the state diagram can be simplified by using a table lookup to recognize reserved words.

Syntax analyzers have two goals: to detect syntax errors in a given program and to produce a parse tree, or possibly only the information required to build such a tree, for a given program. Syntax analyzers are either top-down, meaning they construct leftmost derivations and a parse tree in top-down order, or bottom-up, in which case they construct the reverse of a rightmost derivation and a parse tree in bottom-up order. Parsers that work for all unambiguous grammars have complexity O(n3). However, parsers used for implementing syntax analyzers for programming languages work on subclasses of unambiguous grammars and have complexity O(n).

A recursive-descent parser is an LL parser that is implemented by writing code directly from the grammar of the source language. EBNF is ideal as the basis for recursive-descent parsers. A recursive-descent parser has a subprogram for each nonterminal in the grammar. The code for a given grammar rule is simple if the rule has a single RHS. The RHS is examined left to right. For each nonterminal, the code calls the associated subprogram for that nonterminal, which parses whatever the nonterminal generates. For each terminal, the code compares the terminal with the next token of input. If they match, the code simply calls the lexical analyzer to get the next token. If they do not, the subprogram reports a syntax error. If a rule has more than one RHS, the subprogram must first determine which RHS it should parse. It must be possible to make this determination on the basis of the next token of input.

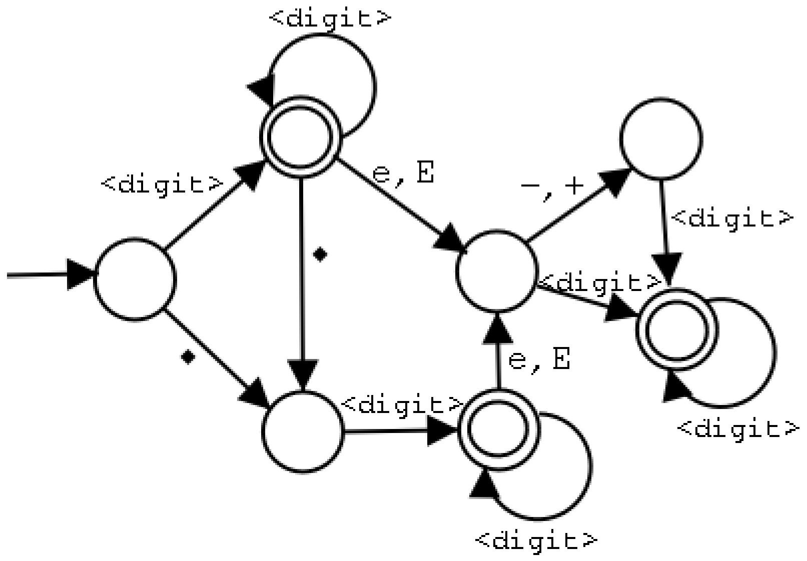
Two distinct grammar characteristics prevent the construction of a recursive-descent parser based on the grammar. One of these is left recursion. The process of eliminating direct left recursion from a grammar is relatively simple. Although we do not cover it, an algorithm exists to remove both direct and indirect left recursion from a grammar. The other problem is detected with the pairwise disjointness test, which tests whether a parsing subprogram can determine which RHS is being parsed on the basis of the next token of input. Some grammars that fail the pairwise disjointness test often can be modified to pass it, using left factoring.

The parsing problem for bottom-up parsers is to find the substring of the current sentential form that must be reduced to its associated LHS to get the next (previous) sentential form in the rightmost derivation. This substring is called the handle of the sentential form. A parse tree can provide an intuitive basis for recognizing a handle. A bottom-up parser is a shift-reduce algorithm, because in most cases it either shifts the next lexeme of input onto the parse stack or reduces the handle that is on top of the stack.

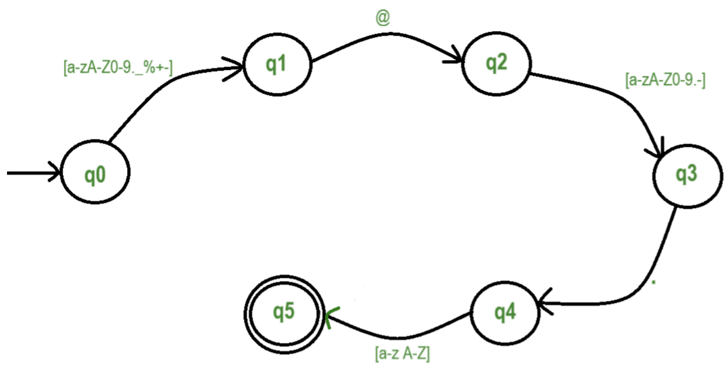
The LR family of shift-reduce parsers is the most commonly used bottom-up parsing approach for programming languages, because parsers in this family have several advantages over alternatives. An LR parser uses a parse stack, which contains grammar symbols and state symbols to maintain the state of the parser. The top symbol on the parse stack is always a state symbol that represents all of the information in the parse stack that is relevant to the parsing process. LR parsers use two parsing tables: ACTION and GOTO.

The ACTION part specifies what the parser should do, given the state symbol on top of the parse stack and the next token of input. The GOTO table is used to determine which state symbol should be placed on the parse stack after a reduction has been done.

Finite State Machine (FSM) to recognize a floating-point number
http://faculty.cooper.edu/smyth/cs225/ch7/fsm.htm



Finite State Machine (FSM) to recognize email address
https://www.geeksforgeeks.org/compiler-design/how-dfa-and-nfa-help-for-tokenization-of-regular-expression/



Review Questions

1. What are three reasons why syntax analyzers are based on grammars?
First using BNF descriptions of the syntax of programs are clear and concise. Second can be used at the direct basis for the syntax analyzer. Third implementations based on BNF are relatively easy to maintain because of their modularity. (Andrew Laboy)

2. Explain the three reasons why lexical analysis is separated from syntax analysis.

1. Efficiency, lexical analysis is often a repetitive process, while syntax analysis is a commonly recursion heavy process. By separating the two processes, specialized and more efficient techniques can be used to do the repetitive lexical analysis, which would otherwise be inefficient to implement inside of syntax analysis, which is recursion based, and is the parser of the lexical analysis (the scanned).
2. Simplicity, having two distinct separate analyzers lowers the complexity of lexical and syntactic analysis
3. Ease of usage, by having syntax analysis separate from lexical analysis, environments which depend on lexical analysis for other purposes than for usage by a syntax analyzer, can package the often simpler lexical analyzer into their environments without having to include the usually recursion heavy syntax analyzer.

(Vincent Comaianni)

3. Define lexeme and token.
A lexeme is the smallest sequence of characters that has meaning, like “if” or “=”. A token is a category or class of lexemes, like “KEYWORD” or ‘ASSIGN_OP”. (Oscar Yang)

4. What are the primary tasks of a lexical analyzer?
Lexical analyzer is a pattern matcher for character strings. (Jackie Lin)

5. Describe briefly the three approaches to building a lexical analyzer.
Lexical analyzer can be built by:
Manual construction of tokens and using a tool to construct a table
Designing a state diagram that describes the tokens and constructing a program that implements it
Designing a state diagram that describes the token and manually construct a table for the state diagram (Jackie Lin)

6. What is a state transition diagram?
dynamic model: Represents changing states of a system (Monika Luchowska)

7. Why are character classes used, rather than individual characters, for the letter and digit transitions of a state diagram for a lexical analyzer?

Character classes are used instead of individual characters to reduce the size and complexity of the state diagram. A lexical analyzer needs to recognize things such as letters(a-z, A-Z) and digits (0-9), if we use individual characters each letter and digit would need its own transition causing a huge mess. Character classes mean fewer transitions and simpler diagrams. (Jeaneth Lliguicota)

8. What are the two distinct goals of syntax analysis?
The two main goals of syntax analysis are to determine whether a program is written according to the grammar of the language and to build a structural representation of the program, such as a parse tree, that shows how the program's parts fit together. (Jahid Hasan)

9. Describe the differences between top-down and bottom-up parsers.
A top-down parser starts from the start symbol and expands the production to match the input, while a bottom-up parser starts from the input and reduces it to the start symbol.(Kaiwen Liu)

10. Describe the parsing problem for a top-down parser.
The goal of the parsing problem is given an input, produce a parse tree that visually represents the syntactical structure of the input string. The top down parser produces a parse tree beginning at the root. This means that you have to decide which production rule to apply while expanding a nonterminal. (Nicholas Roberts)

11. Describe the parsing problem for a bottom-up parser.
A bottom up parser produces a parse tree beginning at the leaves. It decides when and what to reduce, reducing substrings to non terminals. The difficulty comes with identifying the correct reduction at the right time. (Nicholas Roberts)

12. Explain why compilers use parsing algorithms that work on only a subset of all grammars.
The parsing of all grammars is simply too expensive in general. Real compilers require deterministic parsing to avoid ambiguity and achieve efficient procedures. Subsets are easier to implement and reduce compiler complexity. (Tony Chen)

13. Why are named constants used, rather than numbers, for token codes?
Readability, maintainability and reduce errors (Jeremy Haft)

14. Describe how a recursive-descent parsing subprogram is written for a rule with a single RHS.
Example->*Example|Lambda ??? (Jeremy Haft)

15. Explain the two grammar characteristics that prohibit them from being used as the basis for a top-down parser.
Left recursion causes infinite loops and ambiguity/common prefixes which causes prediction failure. (Jigme Topgyal)

16. What is the FIRST set for a given grammar and sentential form?
The set of all terminals that can appear in the start of any string. (Jigme Topgyal)

17. Describe the pairwise disjointness test.
Ensures RHS alternatives have distinct FIRST sets. (Sonu Khadgi)

18. What is left factoring?
Left factoring is a grammar transformation used in compiler design to make the grammar suitable for top-down parsing. (Arsalan Ansari)

19. What is a phrase of a sentential form?
Any subsequence derived from a non-terminal. (Sonu Khadgi)

20. What is a simple phrase of a sentential form?
A phrase with no proper subphrases. (Sonu Khadgi)

21. What is the handle of a sentential form?
The substring must be reduced in the bottom-up parsing. (Sonu Khadgi)

22. What is the mathematical machine on which both top-down and bottom-up parsers are based?
Answer: LL and LR parsers based on **Pushdown Automata**

a) Which is a finite automaton
    i) w/ a stack
    ii) Recognizes Context-Free Languages (Monika Luchowska)

23. Describe three advantages of LR parsers.
It can parse most programming language grammars.
It finds syntax errors quickly.
It is fast and efficient (Adeel Asaf)

24. What was Knuth's insight in developing the LR parsing technique?
They can parse a wider range of grammars than LL ones SPECIFICALLY Left hand side recursion!(Jeremy Haft)

25. Describe the purpose of the ACTION table of an LR parser.
To determine shifts ,reduce, accept or error. (Sonu Khadgi)

26. Describe the purpose of the GOTO table of an LR parser.
To determine the next step after a reduction. (Sonu Khadgi)

27. Is left recursion a problem for LR parsers?
No. Because, they are bottom-up parsers that recognize handles after the input symbols are read rather than expanding production recursively. (Sonu Khadgi)

Chapter 5: Names, Bindings, and Scopes

Summary

Case sensitivity and the use of underscores are the design issues for names.

Variables can be characterized by the sextuple of attributes: name, address, value, type, lifetime, and scope.

Aliases are two or more variables bound to the same storage address. They are regarded as detrimental to reliability but are difficult to eliminate entirely from a language.

Binding is the association of attributes with program entities. Knowledge of the binding times of attributes to entities is essential to understanding the semantics of programming languages. Binding can be static or dynamic. -

Declarations, either explicit or implicit, provide a means of specifying the static binding of variables to types. In general, dynamic binding allows greater flexibility but at the expense of readability, efficiency, and reliability.

Scalar variables can be separated into four categories by considering their lifetimes: static, stack dynamic, explicit heap dynamic, and implicit heap dynamic.

Static scoping is a central feature of ALGOL 60 and some of its descendants. It provides a simple, reliable, and efficient method of allowing visibility of nonlocal variables in subprograms.

Dynamic scoping provides more flexibility than static scoping but, again, at the expense of readability, reliability, and efficiency.

Most functional languages allow the user to create local scopes with let constructs, which limit the scope of their defined names.

The referencing environment of a statement is the collection of all of the variables that are visible to that statement.

Named constants are simply variables that are bound to values only once.

Review Questions

1. What are the design issues for names?
The design issues for names include case sensitivity, handling of reserved words, and the length and form of identifiers, all of which affect readability and writability. (Jahid Hasan)

2. What is the potential danger of case-sensitive names?
It reduces readability and maintainability because different programmers use different naming conventions (eg. myData vs my_data), making it hard to locate errors (Lily Zheng)

3. What is an alias?
An alias is when two or more names refer to the same memory location or data object, causing problems if one changes the data without the other knowing. (Arsalan Ansari).

4. Which category of C++ reference variables always produces aliases?
Lvalue References(Jeremy Haft)

5. What is the l-value of a variable? What is the r-value?
The l-value of a variable is its address, and the r-value is its value. (Jackie Lin)

6. Define binding and binding time.
Binding - association of var / operator w/ attribute; ex: value, type, memory location
Binding time: when association made; ex: language design time - static, execution time - dynamic (Monika Luchowska)

7. After language design and implementation, what are the four times bindings can take place in a program?
After language design and implementation, bindings in a program can take place at compile time, link time, load time, and run time, depending on when program attributes such as types, addresses, and values are associated with program entities. (Jahid Hasan)

8. Define static binding and dynamic binding.
    Static binding is when the binding between a name and an attribute occurs at compile time based on the program's lexical structure whereas dynamic binding occurs when binding is determined at runtime based on the execution context. (Tony Chen)

9. What are the advantages and disadvantages of implicit declarations?
Implicit declaration improves readability and writability, but decreases reliability by allowing misspelled or unintended variables and weakening type checking.(Kaiwen Liu)

10. What are the advantages and disadvantages of dynamic type binding?
- The advantage of dynamic type binding is flexibility. But the disadvantage would be that it reduces reliability and can lead to runtime errors because type errors are detected at run time. (Oscar Yang)

11. Define static, stack-dynamic, explicit heap-dynamic, and implicit heap-dynamic variables. 1)What are their advantages and disadvantages?
Static variables are bound to memory cells before execution begins, and remains bound to the same memory cell throughout execution

a. Advantages- very efficient (because of direct addressing), history sensitive
b. Disadvantages- lacks flexibility, cannot support recursion

2)Stack-Dynamic variables: bindings are created for stack dynamic variables when their deceleration statements are elaborated. Think of it as a temporary workspace when you enter a function that disappears when you leave that function.

c. Advantages -allows recursion, conserves storage
d. Disadvantages- Subprograms cannot be history sensitive, inefficient references (indirect addressing)

3)Explicit heap-dynamic variables: variables whose memory is allocated and deallocated by explicit directives given by the programmer which take effect during execution. Basically the memory is manually requested from the heap by the programmer, and manually released when the programmer is done with the variable

e. Advantages-provides dynamic storage management
f. Disadvantages- inefficient and unreliable

4)Implicit heap dynamic variables: variables whose memory allocation and deallocation are done automatically

g. Advantages- Flexibility
h. Disadvantages- Inefficient because all attributes are dynamic, loss of error detection

(Nicholas Roberts)

12. Define lifetime, scope, static scope, and dynamic scope.
The lifetime of a variable is the time during which it is bound to a particular memory cell. When a variables exist in memory.
The scope of a variable is the range of statements over which it is visible. Where the variable can be seen/used in code.
Static scope is based on program text. It is a reference to a variable that is connected to its declaration by searching the program enclosing scopes, therefore it's determined by where the code is written.
Dynamic Scope is based on the calling sequence of subprograms, not the program text. It is determined by call order. (Jeaneth Lliguicota)

13. How is a reference to a nonlocal variable in a static-scoped program connected to its definition?
By searching the chain of static ancestors, typically implemented at runtime using a static chain. (Jigme Topgyal)

14. What is the general problem with static scoping?
The general problem with static scoping is that it reduces flexibility by binding variable references to their declarations based on program structure at compile time. (Oscar Yang)

15. What is the referencing environment of a statement?
The referencing environment of a statement is the set of all variables that are visible and can be accessed at that point in the program, based on the language's scoping rules. (Jahid Hasan)

16. What is a static ancestor of a subprogram? What is a dynamic ancestor of a subprogram? Static ancestor of a subprogram is lexically enclosing a subprogram. A dynamic ancestor of a program is a caller in the execution sequence. (Sonu Khadgi)

17. What is a block?
G**roup of statements treated as a single unit**

a) Defines scope for variables
b) Controls program flow; ex: if/ while
c) Used visually; ex: "Scratch: coding
d) Structures code
e) manages data
f) Represents sequential code (Monika Luchowska)

18. What is the purpose of the let construct in functional languages?
Create local scopes in functional languages. (Sonu Khadgi)

19. What is the difference between the names defined in an ML let construct from the variables declared in a C block?
An ML let construct creates a value binding (which is immutable), whereas a C block declaration creates a memory location (which is mutable). (Jigme Topgyal)

20. Describe the encapsulation of an F# let inside a function and outside all functions.
In F#, a let inside a function is local to that function, while let outside all function is visible throughout the module (Adeel Asaf)

21. What are the advantages and disadvantages of dynamic scoping?
Advantages: writability and flexibility
Disadvantages: readability and reliability

1. Inability to statically check for references to non local variables
2. Dynamic scoping makes the program hard to read because the calling sequence of sub programs must be known to determine the meaning of references to non local variables
3. There's no way to protect local variables from being accessed to by sub programs because local variables of sub programs are all visible to all other executing sub programs (Andrew Laboy)

22. What are the advantages of named constants?
One of the main advantages of named constants are that they make programs easier to read and understand, as well as prevent accidental changes to fixed values (Adeel Asaf)

Chapter 6: Data Types

Summary

The data types of a language are a large part of what determines that language's style and usefulness. Along with control structures, they form the heart of a language.

The primitive data types of most imperative languages include numeric, character, and Boolean types. The numeric types are often directly supported by hardware.

The user-defined enumeration and subrange types are convenient and add to the readability and reliability of programs.

Arrays are part of most programming languages. The relationship between a reference to an array element and the address of that element is given in an access function, which is an implementation of a mapping. Arrays can be either static, as in C++ arrays whose definition includes the static specifier; fixed stack-dynamic, as in C functions (without the static specifier); fixed heap-dynamic, as with Java's objects; or heap dynamic, as in Perl's arrays.

Most languages allow only a few operations on complete arrays

Records are now included in most languages. Fields of records are specified in a variety of ways. In the case of COBOL, they can be referenced without naming all of the enclosing records, although this is messy to implement and harmful to readability. In several languages that support object-oriented programming, records are supported with objects.

Tuples are similar to records, but do not have names for their constituent parts. They are part of Python, ML, and F#.

Lists are staples of the functional programming languages, but are now also included in Python and C#.

Unions are locations that can store different type values at different times. Discriminated unions include a tag to record the current type value. A free union is one without the tag. Most languages with unions do not have safe designs for them, the exceptions being ML, Swift, and F#.

Pointers are used for addressing flexibility and to control dynamic storage management. Pointers have some inherent dangers: Dangling pointers are difficult to avoid, and memory leakage can occur.

Reference types, such as those in Java and C#, provide heap management without the dangers of pointers.

Enumeration and record types are relatively easy to implement. Arrays are also straightforward, although array element access is an expensive process when the array has several subscripts. The access function requires one addition and one multiplication for each subscript.

Pointers are relatively easy to implement, if heap management is not considered. Heap management is easy if all cells have the same size but is complicated for variable-size cell allocation and deallocation.

Optional type variables are variables that allow a nonvalue to be stored. This allows a program to indicate when a variable currently has no value.

Strong typing is the concept of requiring that all type errors be detected. The value of strong typing is increased reliability. The type equivalence rules of a language determine what operations are legal among the structured types of a language.

Name type equivalence and structure type equivalence are the two fundamental approaches to defining type equivalence.

Type theories have been developed in many areas. In computer science, the practical branch of type theory defines the types and type rules of programming languages. Set theory can be used to model most of the structured data types in programming languages.

Review Questions

1. What is a descriptor?
A descriptor is the collection of attributes of a variable. -Kristopher Garcia

2. What are the advantages and disadvantages of decimal data types?
Advantages: Can precisely store decimal values
Disadvantages: Restricted range (Since exponents are not implemented), Wasted memory due to implementation method (binary coded decimal (BCD)) -Kristopher Garcia

3. What are the design issues for character string types?
One of the main issues for character string types would be whether strings are built-in types or just arrays of characters. (Adeel Asaf)

4. Describe the three string length options.
Static length is fixed before execution and **never** changed.
Limited dynamic length can be changed, but only up to maximum. Max size is set before execution
Dynamic Length can be changed freely at runtime and no fixed upper bound.(Jeaneth Lliguicota)

5. Define ordinal, enumeration, and subrange types.
Ordinal: A data-type whose value can be ordered and counted such as integers, character, boolean, etc.
Enumeration: user-defined ordinal type that consists of finite sets of named values such as {red, green, blue}.
Subrange types: an ordered subset of an existing ordinal type with restricted range of values such as integers from 1 to 10. (Sonu Khadgi)

6. What are the advantages of user-defined enumeration types?
It improves readability and reliability while being able to stay efficient. (Vegus Tao)

7. In what ways are the user-defined enumeration types of C# more reliable than those of C++?
They have type safety and prevention of implicit conversions. In C# enums aren't thought of as equivalent to int values. They are their own thing. You must force a type cast explicitly! (Jeremy Haft)

8. What are the design issues for arrays?
Design issues for arrays are indexing, binding, allocation, memory layout, operations, and initialization. (Jigme Topgyal)

9. Define static, fixed stack-dynamic, fixed heap-dynamic, and heap-dynamic arrays. What are the advantages of each?
Static arrays are bound at compile time and have a fixed size and lifetime, which makes them efficient and fast to access. Fixed stack-dynamic arrays are allocated at runtime when a subprogram begins execution but have a fixed size, offering flexibility with low overhead. Fixed heap-dynamic arrays are allocated from the heap with a size fixed at creation, allowing more flexibility in size than stack arrays. Heap-dynamic arrays can change size during execution, providing maximum flexibility for dynamic data structures. (Jahid Hasan)

10. What happens when a nonexistent element of an array is referenced in Perl?
The element is automatically created and initialized. (Sonu Khadgi)

11. How does JavaScript support sparse arrays?
A sparse array is an array (data structure) where nearly all the elements are 0 or a default value. Keeps repeating values from taking up precious space! Javascript implements them as specialized objects (Jeremy Haft).

12. What languages support negative subscripts?
Languages that support negative subscripts are, APL and Fortran. (Arsalan Ansari)
Python s[-1] means the last character in the string, s[-2] is the second to last…
.
13. What languages support array slices with stepsizes?
Languages that support array slices with stepsizes are Python, Fortran, and Ada.(Kaiwen Liu)

14. What is an aggregate constant?
A vector constant whose value is not changed during entire programs execution. (Andrew Laboy)

15. Define row major order and column major order.
Row-major order stores the elements of each row in consecutive memory locations, with rows placed one after another. Column-major order stores the elements of each column contiguously, with columns placed one after another in memory. (Jahid Hasan)

16. What is an access function for an array?
An access function for an array is the mapping of its base address and a set of index values to the address in the memory of the element specified by the index value (Andrew Laboy)

17. What are the required entries in a Java array descriptor, and when must they be stored (at compile time or run time)?
- The required entries in a Java array descriptor is length of array, type of elements, and the pointer to the actual data/elements. The array length is stored at runtime, the element type is stored at compile time, and the pointer is stored at runtime. (Oscar Yang)

18. What is the structure of an associative array?
Associative array is an unordered collection of data elements that are indexed by an equal number of values called Keys. (Andrew Laboy)

19. What is the purpose of level numbers in COBOL records?

Answer: level numbers in COBOL define data hierarchy

a) Grouping items into records; ex:
    i) 01-highest level number
b) Identifying special data types; ex:
    i) RENAMES - 66
    ii) Independent items - 77
    iii) Condition names - 88
c) Structures complex data
    i) Shows nesting
    ii) Relationships bt/wn data fields (Monika Luchowska)

To define the hierarchical structure. Lower numbers are boarder while higher numbers are more specific - (Dariel Urena)

20. Define fully qualified and elliptical references to fields in records.

21. What is the primary difference between a record and a tuple?
The primary difference is how the elements are accessed and identified. Records elements are accessed by field names where each field has a name and type, but tuples elements are accessed by their index. (Oscar Yang)

22. Are the tuples of Python mutable?

- No, tuples in Python are immutable. Once a tuple is created, its elements cannot be changed, added, or removed, which makes tuples safer to use for fixed collections of data. (Jahid Hasan)

23. What is the purpose of an F# tuple pattern?
The purpose of an F# pattern is to decompose a tuple into its individual components so each value can be bound to a separate name. (Sonu Khadgi)

24. In what primarily imperative language do lists serve as arrays?
In LISP, lists serve as the primary data structure that takes the role of arrays.(Kaiwen Liu)

25. What is the action of the Scheme function CAR?
CAR returns the first element of a non-empty list(Nicholas Roberts)

26. What is the action of the F# function tl?
The action of the F# function tl returns the tail of the list, which is the list without the first element. And if the tl is applied to an empty list, it results in a run-time error. (Sonu Khadgi)

27. In what way does Scheme's CDR function modify its parameter?

- It doesn't modify its parameter. In Scheme, cdr is non-destructive: it returns the tail of a pair/list (everything after the first element) without changing the original list. (azmain a)

28. On what are Python's list comprehensions based?
Python's list comprehensions are based on set-builder notation from mathematics.
E.g.:
num=[1,2,3]
odd=[n for n in num if n%2==1]
It will generate an odd number list(odd[1,3])
And in Maths, it means odd = { n | n ∈ num, n is odd }
(Cheuk Yiu Sung)

29. Define union, free union, and discriminated union.

- A union is a data structure in which all members share the same memory location, but only one member can store a value at a time. A free union is a union in which no tag or indicator is stored to identify the currently active member and the programmer must manually keep track of which field is valid. A discriminated union is a union which includes a discriminator (tag field) to indicate which member currently holds a valid value. (Oscar Yang)

30. Are the unions of F# discriminated?
Yes, because a) each case has distinct identifier
b) each case has diff data types + data amounts
c) the pattern-matching “match” expression handles every case (Monika Luchowska0

31. What are the design issues for pointer types?
The design issues for pointer types include deciding what objects pointers can reference, whether pointers are typed, if pointer arithmetic is allowed, the memory management strategy, and how to handle null or dangling pointers. (Oscar Yang)

32. What are the two common problems with pointers?
Dangling pointers where you delete the object is pointing to and now the pointer keeps the old address but the pointer is either invalid or points to an unrelated object. Another common problem is memory leaks where you assign a pointer a new value and now the old value cannot be accessed or freed, taking up space in the memory. (Vegus Tao)

33. Why are the pointers of most languages restricted to pointing at a single type variable?
The pointers of the most languages are restricted to pointing at a single type variable to enable effective type checking and prevent unsafe memory access, thereby improving program reliability. (Sonu Khadgi)

34. What is a C++ reference type, and what is its common use?
A reference type is another known name for a variable, it must be initialized, is not an object, cannot be null or reseated, and is automatically dereferenced. Common use is for pass by reference to avoid copying the original data twice. -Kristopher Garcia

35. Why are reference variables in C++ better than pointers for formal parameters?
Reference variables in C++ are better than pointers for formal parameters because they provide implicit dereferencing, making the code easier to read and write, and they cannot be null, which improves safety and reliability. This reduces common pointer-related errors while still allowing efficient pass-by-reference behavior. (Jahid Hasan)

36. What advantages do Java and C# reference type variables have over the pointers in other languages?
Reference type variables have an advantage in safety as the user cannot use arithmetic on references unlike pointers which prevents bugs such as buffer overflowing. The references are also implicitly dereferenced and deallocated by their respective GC’s. (Tony Chen)

37. Describe the lazy and eager approaches to reclaiming garbage.
Lazy does not immediately clear up space for memory, waiting until it starts running out of memory to clear it. Eager clears up memory right away. (Dariel Urena)

38. Why wouldn’t arithmetic on Java and C# references make sense?
In Java and C#, the languages abstract away the direct memory manipulation standard in languages like C/C++. Java/C# also enforce type safety and use a garbage collector which would change the address of a pointer once the GC runs. (Weitong Chen)

39. What is a compatible type?
Answer: Similar enough types, used interchangeably; ex: int and unsigned int in C (Monika Luchowska)

40. Define type error.

A type error occurs when an operation is applied to data of an incompatible or incorrect type, such as adding a number to a string or assigning a value to a variable of the wrong type. (Jahid Hasan)

41. Define strongly typed.
Strongly typed language reinforces the consistency of the variables and what data type it can interact with. (Jackie Lin)

42. Why is Java not strongly typed?
Java is not strongly typed because it allows for type errors to happen at runtime and not compile time. (Joseph Pasqualicchio)

43. What is a nonconverting cast?
Operation that changes data type associated w/ a value
w/o altering underlying bit representation
used in low-level programing Ex: memory allocation algo's that manage heap
No runtime code execution for conversion (Monika Luchowska)

44. What languages have no type coercions?
Some languages like Haskell do not have type coercions to improve type safety and reliability. (Joseph Pasqualicchio)

45. Why are C and C++ not strongly typed?
In C/C++, due to their support for implicit type conversions, the use of explicit type casting to bypass the type system, and the ability to directly manipulate memory via pointers, leading to potential type confusion, it is not considered "strongly typed". (Weitong Chen)

46. What is name type equivalence?
Names type equivalence mean when to types are the same only if they have same name (Adeel Asaf)

47. What is structure type equivalence?
It means two types are considered the same if they have the same structure, even if they have different names. (Adeel Asaf)

48. What is the primary advantage of name type equivalence?
The primary advantage of name type equivalence is it provides stronger type safety. In name type equivalence two types are the same if they have the same name. So if there was int age and int salary, it would catch that if you did age = salary, even if both are ints. (Oscar Yang)

49. What is the primary disadvantage to structure type equivalence?
Structure type equivalences are more flexible but much harder to implement
(Nicholas Roberts)

50. For what types does C use structure type equivalence?
In C, structure type equivalence applies to struct, union, and enum types and is determined by tag names rather than the fields' layout. (Kaiwen Liu)

51. What set operation models C's struct data type?
C's struct data type is modeled by the Cartesian product of its component types. (Sonu Khadgi)

Chapter 7: Expressions and Assignment Statements

Summary

Expressions consist of constants, variables, parentheses, function calls, and operators. Assignment statements include target variables, assignment operators, and expressions.
The semantics of an expression is determined in large part by the order of evaluation of operators. The associativity and precedence rules for operators
in the expressions of a language determine the order of operator evaluation in those expressions. Operand evaluation order is important if functional side effects are possible. Type conversions can be widening or narrowing. Some narrowing conversions produce erroneous values. Implicit type conversions, or coercions, in expressions are common, although they eliminate the error-detection benefit of type checking, thus lowering reliability.
Assignment statements have appeared in a wide variety of forms, including conditional targets, assigning operators, and list assignments.

Review Questions

1. Define operator precedence and operator associativity.
Operator precedence determines which operator gets evaluated first similar to how the order of operations works in math., The operator associativity is regarding when the same operator is used, which one takes priority first (right to left as in like a - b - c) or (a = b = c). (Joseph Pasqualicchio)

2. What is a ternary operator?
A ternary operator takes three operands and works as a compact if else statement, is commonly used like:
(truth value) ? (if true take this) : (if false take this) with the ? serving as an if and the : serving as an or depending on the result of the truth value (Joseph Pasqualicchio)

3. What is a prefix operator?
A prefix operator is one that is used before an operand, like ++variable or –variable, where we put the operator before the variable type and the operator will influence the result (Joseph Pasqualicchio)

4. What operator usually has right associativity?
The assignment operator a + b **=** c (Joseph Pasqualicchio)

5. What is a nonassociative operator?
A non-associative operator is one which cannot be chained / used multiple times in a single expression, unless wrapped in parentesis. An example of invalid use is: a < b < c, but (a < b) && (b < c) is a valid use. (Joseph Pasqualicchio)

6. What associativity rules are used by APL?
- APL uses right-to-left associativity for evaluating expressions and does not rely on traditional operator precedence rules. Instead, expressions are evaluated strictly from right to left unless parentheses are used to explicitly control the order of evaluation. (Jahid Hasan)

7. What is the difference between the way operators are implemented in C++ and Ruby?
- In C++, operators are built into the language and can be overloaded using special syntax, with most decisions made at compile time. In Ruby, operators are methods, so using an operator is simply a method call handled at runtime. (Jahid Hasan)

8. Define functional side effect.

A functional side effect occurs when there is a change in function state different than its returned value. An example of this could be (Joseph Pasqualicchio):

```
var x = 0
fun set() {
 x += 3 // this is a functional side effect
 return x
}
```

9. What is a coercion?   Coercion is an implicit type conversion that occurs during expression evaluation. The language automatically changes the value’s type. (Jeaneth Lliguicota)

10. What is a conditional expression?
A conditional expression is an expression that evaluates to one of two values depending on a Boolean condition. (Arsalan Ansari).

11. What is an overloaded operator?
Operator overloading is a feature where a language allows you—the programmer—to redefine how operators like +, -, or += behave for your own custom data types. Many languages do not allow this, but it is a powerful feature in Object-Oriented Programming (OOP). It lets you create a custom class (like a "Complex Number" or "Vector") that can use standard math operators in a way that makes sense for that specific class.（Cheuk Yiu Sung）

12. Define narrowing and widening conversions.
Narrowing is when you convert a data type to a type where it might not be able to hold all possible values (double to int). Widening is when you convert a data type to a type where it can hold all possible values and more (int to double) - (Dariel Urena)

13. In JavaScript, what is the difference between == and ===?
- a) **== “Loose Equality”**
  - i) checks for same value
  - ii) Performs type coercion before comparison
    - 1) Ex: “5” == 5 -> true
      - (a) Have same value, 5!
- b) **=== “Strict Equality”**
  - i) Checks if two operators have same:
    - 1) Value
    - 2) type
  - a) comparison w/o type conversion
    - i) Ex: “5” === 5 -> false
      - 1) String and number are different! (Monika Luchowska)

14. What is a mixed-mode expression?
Its when you add/operate with 2 variables of different types (Jeremy Haft)

15. What is referential transparency?
Referential transparency means an expression can be replaced by its value without affecting the program's behavior. An example of this is:

```
def square(x):
   return x * x
y = square(3) + square(3)
//we can replace square(3) with 9 without changing the program
y = 9 + 9
```

(Kaiwen Liu)

16. What are the advantages of referential transparency?
Programs are easier to understand and reason about as well as programs are easier to test, debug and optimize. (Adeel Asaf)

17. How does operand evaluation order interact with functional side effects?
Readability and portability. The program result is ambiguous since it depends on the order evaluation which creates a problem known as non-determinism. Also the program might work on one compiler but break on another. (Jigme Topgyal)
18. What is short-circuit evaluation?
An expression in which the result is determined without running the rest of the operands and operators.(Jackie Lin)

19. Name a language that always does short-circuit evaluation of Boolean expressions. Name one that never does it.
Java always uses short circuit evaluation: Stopping evaluation when result is known
C never short-circuits: --Evaluating all operands (Monika Luchowska)

20. How does C support relational and Boolean expressions?
- C supports relational expressions using operators such as <, >, <=, >=, ==, and !=, which evaluate to integer values (1 for true, 0 for false). Boolean expressions are formed using logical operators &&, ||, and !, but C does not have a built-in Boolean type in older standards, so integers are used to represent truth values. (Jahid Hasan)

21. What is the purpose of a compound assignment operator?
Concise syntax; automatic data type conversion is performed at the underlying level during the operation. (Cheuk Yiu Sung)

22. What is the associativity of C's unary arithmetic operators?
Right to Left. E.g. *-7, ++x* . Unary operator means an operator follows by only one operand (Cheuk Yiu Sung)

23. What is one possible disadvantage of treating the assignment operator as if it were an arithmetic operator?
One possible disadvantage of treating the assignment operator as if it were an arithmetic operator is it can lead to unintentional logical errors, where an assignment is performed when a comparison was needed. (Jigme Topgyal)

24. What two languages include multiple assignments?
Python and Ruby. Both allow code like "a, b = 1, 2" (Oscar Yang)

25. What mixed-mode assignments are allowed in Java?
In Java, mixed-mode assignments are only allowed if the assignment is a lossless conversion such as int to double and not the other way around. (Tony Chen)

26. What mixed-mode assignments are allowed in ML?
None of them because error detection is reduced when mixed mode expressions are allowed. (Andrew Laboy)

27. What is a cast?
A cast is an explicit type conversion. The system looks at what specific data type we are using and then changes it to be the one we want it to be (Jeaneth Lliguicota)