HW 1

Q1:

2.2:

- ((A) (B))

- (A (B (C)))

- (((A) (B)) (C))

2.4:

- (BOWS ARROWS FLOWERS CHOCOLATES)

2.6:

1- () -> (NIL)

2- (()) 🡪 NIL

3- ((())) 🡪 (NIL)

4- (() ()) 🡪 (NIL NIL)

5- (() (())) 🡪 (NIL (NIL))

2.13:

|  |  |
| --- | --- |
| Function | Result |
| CAR | ((FUN)) |
| CADR | (IN THE) |
| CADDR | (SUN) |
| CAAR | (FUN) |
| CAADR | IN |
| CADAR | THE |
| CADDAR | THE |
| CADDDR | SUN |

2.15:

|  |  |
| --- | --- |
| Function | Result |
| CAR | (A B) |
| CDDR | ((E F)) |
| CADR | (C D) |
| CDAR | (B) |
| CADAR | B |
| CDDAR | (D) |
| CAAR | A |
| CADDR | (E F) |
| CDADDR | F |

2.16:

CAAR of (FRED NIL) gives NIL, because the CAR of (FRED NIL) is NIL.

Q2:

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?

A background in language design helps a programmer better understand the strengths, limitations, and appropriate uses of different programming languages. It also enables them to write more efficient and readable code, as they can leverage the language's features effectively.

1. How can knowledge of programming language characteristics benefit the whole computing community?

It allows developers to choose the right language for the right task, fostering innovation and efficiency. A deeper understanding promotes interoperability, improves software quality, and drives advancements in compiler and interpreter design.

1. What programming language has dominated scientific computing over the past 60 years?

Fortran has dominated scientific computing due to its focus on numerical computation and high-performance execution.

1. What programming language has dominated business applications over the past 60 years?

COBOL has dominated business applications because of its ability to handle large volumes of data processing and its alignment with business requirements.

1. What programming language has dominated artificial intelligence over the past 60 years?

LISP has been a dominant language in artificial intelligence due to its support for symbolic computation and rapid prototyping.

1. In what language is most of UNIX written?

Most of UNIX is written in C.

1. What is the disadvantage of having too many features in a language?

Too many features can make the language complex, reducing readability, writability, and maintainability. It can also overwhelm new learners and lead to inconsistent or inefficient code.

1. How can user-defined operator overloading harm the readability of a program?

If overloading is done inconsistently or without clear intent, it can make the code ambiguous and harder to understand, as the meaning of operators may vary across different contexts.

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

Arrays and pointers in C are not entirely orthogonal. For example, arrays can decay into pointers in certain contexts, but they are not interchangeable in others, leading to inconsistencies.

1. What language used orthogonality as a primary design criterion?

ALGOL 68 prioritized orthogonality in its design, ensuring that language features could be combined consistently.

1. What primitive control statement is used to build more complicated control statements in languages that lack them?

The GOTO statement is often used as a primitive control statement to build more complex control structures.

1. What does it mean for a program to be reliable?

A program is reliable if it performs its intended functions under all expected conditions without failure, including handling errors gracefully.

1. Why is type checking the parameters of a subprogram important?

Type checking ensures that the arguments passed to a subprogram are compatible with its expected types, reducing errors and increasing program reliability.

1. What is aliasing?

Aliasing occurs when two or more variables reference the same memory location. This can lead to unexpected side effects and bugs if one variable modifies the shared data.

1. What is exception handling?

Exception handling is a mechanism for detecting, managing, and recovering from errors or exceptional conditions that occur during program execution, allowing the program to continue running or fail gracefully.

1. Why is readability important to writability?

Readable code is easier to understand, maintain, and modify. This makes it easier to extend or adapt the program, improving writability and reducing the likelihood of errors.

1. What two programming language deficiencies were discovered as a result of the research in software development in the 1970s?
   1. Lack of data abstraction: Many early programming languages did not allow programmers to define and encapsulate data structures and their associated operations.
   2. Inadequate control structures: Many languages lacked structured programming constructs (e.g., while, for loops), leading to unstructured and hard-to-maintain code, often reliant on GOTO.
2. What are the three fundamental features of an object-oriented programming language?
   1. Encapsulation: Bundling of data and methods into objects.
   2. Inheritance: Ability for one class to inherit the properties and methods of another.
   3. Polymorphism: Ability to define a common interface with multiple implementations, enabling objects to be treated as instances of their parent class.
3. What language was the first to support the three fundamental features of object-oriented programming?

Smalltalk was the first language to fully support encapsulation, inheritance, and polymorphism.

1. What is an example of two language design criteria that are in direct conflict with each other?
2. Readability vs. Writability: A language designed for concise code (high writability) may sacrifice readability. For example, Perl prioritizes writability but can be harder to read.
3. Efficiency vs. Portability: A language optimized for hardware-specific efficiency (e.g., assembly language) may not be portable across platforms.
4. What are the three general methods of implementing a programming language?
5. Compilation: Translating source code into machine code before execution.
6. Pure Interpretation: Directly executing the source code line by line.
7. Hybrid Implementation: Combining both compilation and interpretation, e.g., compiling to intermediate code (like bytecode) and then interpreting it.
8. Which produces faster program execution, a compiler or a pure interpreter?

A compiler produces faster program execution because the source code is translated into optimized machine code prior to execution, whereas a pure interpreter incurs runtime overhead by interpreting the code line by line.

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

Ease of debugging: Errors can be caught and reported immediately as the code is executed line by line.

Flexibility: Programs can be modified and tested interactively without requiring recompilation.

Platform independence: Pure interpreters can execute the same source code on different platforms as long as the interpreter is available.