Foundations of Programming

1) What is a transpiler? Can you name a popular programming language that uses a transpiler?

A) A transpiler, also known as a source-to-source compiler, is a program that translates source code written in one high-level programming language into equivalent source code in another high-level programming language. Unlike a traditional compiler that translates high-level code into low-level machine code, a transpiler maintains a similar level of abstraction between the input and output languages. The output code from a transpiler is still human-readable and typically requires further compilation or interpretation to be executed.

A popular programming language that uses a transpiler is TypeScript. TypeScript code is transpiled into JavaScript, allowing developers to leverage features like static typing and interfaces that are not natively available in standard JavaScript, while ensuring compatibility with various JavaScript environments.

2) What are the executable container formats used in Windows 11 and MacOS?

A) Both Windows 11 and MacOS execute binary executable files, although they are in different formats. The big difference in what a user sees if using the GUI is that on Windows, you usually see the .exe executable file, which resides in a folder that contains other files and information that the program uses. On MacOS , all those other files, along with the primary executable, are contained in a "bundle" (really a folder) with the extension .app. Double clicking on a .app bundle will execute the internal executable and load your application.

Outside of the GUI, in the actual file structure, Mac and Windows systems are really quite similar; these differences are mostly a question of how the system presents things to the user for manipulation in the GUI

3) Do you know of any notation used to describe the syntax of programming languages? Try to express an email address in one of these notations.

BNF stands for Backus Naur Form notation. It is a formal method for describing the syntax of programming language which is understood as Backus Naur Formas introduced by John Bakus and Peter Naur in 1960. BNF and CFG (Context Free Grammar) were nearly identical.

4) **Do you think a program without functions will be faster than a program for the same logic implemented with functions? Why?**

Theoretically, a program without functions, in which all logic is contained within the main execution flow, might perform minusculely better than a functionally comparable program with functions. This is brought on by the overhead involved in calling functions, which includes things like managing the call stack, moving arguments onto the stack, and executing jump instructions to and from the function's code block.

However, this theoretical advantage is often negligible in modern computing environments for several reasons

1) Compiler Optimizations
2) CPU Caching
3) Readability and Maintainability

5) **Can errors in call stack or stack frame lead to program errors like wrong functionality or program crashes?**

Yes, program issues such as crashes and incorrect functionality can be directly caused by call stack or stack frame errors. When a software tries to use more memory on the call stack than is available, it is known as a "stack overflow," and it frequently happens as a result of excessive or infinitely recursive function calls. The program may crash if the stack space is used up. Furthermore, if data within a stack frame is corrupted—for example, by using improper return addresses or corrupted local variables—the program may execute unwanted instructions or operate on inaccurate data, which could result in incorrect functionality or unexpected behavior.

6) **Do you think security related vulnerabilities could be associated with call stacks? Can you think of an example for this?**

It is true that call stacks might be linked to security flaws, especially when it comes to memory-related errors. A key data structure in computer programming, the call stack holds details about a program's running subroutines, such as function arguments, return addresses, and local variables. Exploiting call stack vulnerabilities can result in a number of security lapses.

A buffer overflow vulnerability is a well-known illustration. This happens when a program tries to write more data than the stack's allotted buffer size. This has the ability to rewrite nearby memory locations, including the call stack's return address. By using shellcode, an attacker can create malicious input that replaces the return address with the address of their own malicious code**.**

7) **How will the set up of the call stack work for a recursive function call?**

For every recursive invocation of a recursive function, a new stack frame is pushed into the call stack. The local variables, parameters, and return address—the place in the calling function's code where execution should continue when the current function finishes—are among the crucial details for that particular function call that are contained in this stack frame. New frames are progressively added to the top of the stack as the recursive calls proceed, therefore stopping the preceding calls' execution. The function starts to return and pops the matching stack frame off the stack when a base case is reached within the recursion. With its own set of local variables and parameters, this procedure then permits the previously paused function call (the one whose frame is currently at the top of the stack) to continue where it left off. Eventually, it will return, popping its frame. Until all recursive calls have finished and their frames have been deleted, this unwinding process keeps going until the original caller regains control.

8) **Can a call stack grow indefinitely? What happens if a function is called recursively too many times?**

No, a call stack can't keep growing. The operating system allots a certain amount of RAM to the call stack, which holds data about active function calls. A function will use up this allotted memory if it is invoked repeatedly too frequently without reaching a base case to end the recursion. A stack overflow is the phrase for this situation, which results in a runtime error and an unexpected program crash or termination.

9) **What does it mean to say that functions or expressions can have "side effects"?**

A function or expression is said to have "side effects" in programming if it alters something outside of its own scope, like a file, a global variable, or the state of another object, in addition to returning a value. In essence, it is an action that modifies the state of the program in addition to the function or expression's immediate input and output.

Examples of side effects:

- **Modifying a global variable:**

  If a function changes the value of a variable defined outside its own scope, that's a side effect.

- **Writing to a file:**

  Saving data to a file or reading from a file are examples of input/output operations that constitute side effects.

- **Altering object state:**

If a function changes the properties or state of an object, it's not directly working with, it's considered a side effect.

10) **How can one stop the execution of a loop before it is completed? Illustrate the difference between 'break' and 'continue' with an example**.

The execution of a loop can be stopped or altered before its natural completion using control flow statements like break and continue.

break statement:

The break statement immediately terminates the loop in which it is contained. Program control then resumes at the statement immediately following the loop.

continue statement:

The continue statement skips the current iteration of the loop and proceeds to the next iteration. It does not terminate the loop entirely but rather bypasses the remaining code within the current iteration.

11) Write the pseudo code to find the sum of the cubes of the first 9 positive integers.

```
ALORITHM SumOfCubesOfFirstNineIntegers
        SET total_sum to 0
        FOR each_number FROM 1 T0 9
                SET cube_of_number TO each_number * each_number * each_number
                ADD cube_of_number TO total_sum
        END FOR
        DISPLAY total_sum

END ALORITHM
```

Execution Java: https://www.programiz.com/online-compiler/87DpEYZRNOy4b

ExecutionTypeScript: https://www.programiz.com/online-compiler/5gyoPkZx3umW4

12) Write the pseudo code to check if a given 4-digit square number will yield another square number by adding 1 to each digit. For example, if 2025 is the input, adding 1 to each digit yields 3136.

```
FUNCTION isPerfectSquare(n):
   sqrt_n ← integer(sqrt(n))
```

```
    RETURN (sqrt_n * sqrt_n == n)

FUNCTION addOneToEachDigit(n):
    result ← ""
    FOR each digit d in n:
        result ← result + (d + 1)
    RETURN integer(result)

FUNCTION checkSquare(n):
    IF NOT isPerfectSquare(n):
        PRINT "Input is not a square"
        RETURN

    new_num ← addOneToEachDigit(n)

    IF isPerfectSquare(new_num):
        PRINT n, "→", new_num, "both are squares"
    ELSE:
        PRINT n, "→", new_num, "second is not a square"
```

Execution Java: https://www.programiz.com/online-compiler/5AF4DWPIQsM0m

Execution TypeScript: https://www.programiz.com/online-compiler/0wjOHRIpmTVkf

Github: https://github.com/siva2555/Matrimorphosis.git