Subroutines and Control Abstraction

1. What is a subroutine calling sequence? What does it do? What is meant by the subroutine prologue and epilogue?

2. How do calling sequences typically differ in older (`CISC`) and newer (`RISC`) instruction sets?

3. Describe how to maintain the static chain during a subroutine call.

4. What is a display? How does it differ from a static chain?

5. What are the purposes of the stack pointer and frame pointer registers? Why does a subroutine often need both?

6. Why do modern machines typically subroutine parameters in registers rather than on the stack?

7. Why do subroutine calling conventions often give the caller responsibility for saving half the registers and the callee responsible for saving the other half?

8. If work can be done in either the caller or the callee, why do we typically prefer to do it in the callee?

9. Why do compilers typically allocate space for arguments in the stack, even when they pass them in registers?

10. List the optimizations that can be made to the subroutine calling sequence in important special cases (e.g., leaf routines).

11. How does an in-line subroutine differ from a macro?

12. Under what circumstances is it desirable to expand a subroutine in-line?

13. What is the difference between formal and actual parameters?

14. Describe four common parameter-passing modes. How does a programmer choose which one to use when?

15. Explain the rationale for `READONLY` parameters in Modula-3.

16. What parameter mode is typically used in languages with a reference model of variables?

17. Describe the parameter modes of Ada. How do they differ from the modes of other modern languages?

18. Give an example in which it is useful to return a reference from a function in C++.

19. What is an r-value reference? Why might it be useful?

20. List three reasons why a language implementation might implement a parameter as a closure.

21. What is a conformant (open) array?

22. What are default parameters? Why are they useful?

23. What are named (keyword) parameters? Why are they useful?

24. Explain the value of variable-length argument lists. What distinguishes such lists in Java and C# from their counterparts in C and C++?

25. Describe three common mechanisms for specifying the return value of a function. What are their relative strengths and drawbacks?

26. Describe three ways in which a language may allow programmers to declare exceptions.

27. Explain why it is useful to define exceptions as classes in C++, Java, and C#.

28. Explain te behavior and purpose of a `try...finally` construct.

29. Describe the algorithm used to identify an appropriate handler when an exception is raised in a language like Ada or C++.

30. Explain how to implement exceptions in a way that incurs no cost in the common case (when exceptions don't arise).

31. How do the exception handlers of a functional language like ML differ from those of an imperative language like C++?

32. Describe the operations that must be performed by the implicit handler for a subroutine.

33. Summarize the shortcomings of the `setjump` and `longjump` library routines of C.

34. What is a `volatile` variable in C? Under what circumstances is it useful?

35. What was the first high-level programming language to provide coroutines?

36. What is the difference between a coroutine and a thread?

37. Why doesn't the transfer library routine need to change the program counter when switching between coroutines?

38. Describe three alternative means of allocating coroutine stacks. What are their relative strengths and weakneses?

39. What is a cactus stack? What is its purpose?

40. What is discrete even simulation? What is its connection with coroutines?

41. What is an event in the programming language sense of the word?

42. Summarize the two main implementation strategies for events.

43. Explain the appeal of anonymous delegats (C#) and anonymous inner classes (Java) for handling events.

44. Describe how we access an object at lexical nesting level k in a language implementation based on displays.

45. Why isn't the display typically kept in registers?

46. Explain how to maintain the display during subroutine calls?

47. What special concerns arise when creating closures in a language implementation that uses displays?

48. Summarize the tradeoffs between displays and static chains. Describe a program for which displays will result in faster code. Describe another for which static chains will be faster.

49. For each of our three case studeis, explain which aspects of the calling sequence and stack layout are dictated by the hardware, and which are a matter of software convention.

50. Why don't `LLVM` and `gcc` restore caller-saves registers immediately after a call?

51. What is a subroutine closure trampoline? How does it differ from the usual implementation of a closure described in Section 3.6.1? What are the comparative advantages of the two alternatives?

52. Explain the circumstances under which a subroutine needs a frame pointer (i.e., under which access via displacement addressing from the stack pointer will not suffice).

53. Under what circumstances must an argument that was passed in a register also be saved into the stack?

54. What is the purpose of the "red zone" on x86-64?

55. What are register windows? What purpose do they serve?

56. Which commercial instruction sets include regiser windows?

57. Explain the concepts of register window overflow and underflow.

58. Why are register windows a potentional problem for multithreaded programs?

59. What is call by name? What language first provided it? Why isn't it used by the language's descendants?

60. What is call by need? How does it differ from call by name? What modern languages use it?

61. How does a subroutine with call-by-name parameters differ from a macro?

62. What is a thunk? What is it used for?

63. What is Jensen's device?

64. Describe the "obvious" implementation of iterators using coroutines.

65. Explain how the state of multiple active iterators can be maintained in a single stack.

66. Describe the transformation used by C# compilers to turn a true iterator into an iterator object.

67. Summarize the computational model of discrete event simulation. Explain the significance of the time-based priority queue.

68. When building a discrete event simulation, how does one decide which things to model with coroutines, and which to model with data structures?

69. Are all inactive coroutines huaranteed to be in the priority queue? Explain.