PL 92 qual

1. Function passing
    a. Functions can only be called is the least expressive option. It is the easiest implementation because you can still obey a strict stack discipline, and you don't have to worry about closures.
    b. Functions as arguments is the second most expressive option. It is more difficult to implement than a), but stil not bad. You can follow a strict stack discipline, but you will need function closures. The links in the activation records can form a tree. An example is the map function.
    c. Functions can be returned (and passed as arguments) is the most expressive option. It is also the most difficult to implement. You need closures, plus you can no longer obey a stack discipline. An example of this is an interpreter taking user input and running the program

    C avoids the tradeoff by using function pointers. These can be passed or returned, but don't include the environment of the function, just a pointer to code address.
2. Structures and Classes in C++
    a. p.x = 1, p.y = 2
    b. p.x = 1, p.y = 0; q.x = 1, q.y = 1, q.dir = 0.5
       p.move will execute the move code from the pt class. Naieve programmer would expect p.move to call move code from class dp. Implementationally, assignment to p does not change the vtable of p.
    c. p-> move will execute the move code from class dp. The pointer changes from referencing a pt object to referencing a class dp object, and thus gets the associated vtable.
    d. The big reason is dynamic lookup (from the vtable)
3. Futures…
    a. O(n)
    b.
        i. Can't modify any values used in later parts of the statement
        ii. Can't read any values modified in later parts of the statement
    c. You could exit the try block early, thus leaving the scope of your handlers, only to have an exception raise later and not be able to catch it. It is also difficult for the programmer to get it right.