08장 서브루틴과 제어 추상화

【8.2.1】 디스플레이

예 8.59
디스플레이를 사용한 비지역 접근

책에서 언급한 것처럼 디스플레이는 배열로의 정적 체인을 내장시키는 것이다. 디스플레이의 j번째 원소는 가장 최근에 활성화된 어휘적으로 j만큼 중첩된 수준에 있는 서브루틴의 프레임으로의 참조를 포함한다. 그에 따라 디스플레이의 첫 원소는 최근에 활성화된 루틴에 도착할 때까지 메인 프로그램에 직접적으로 중첩된 일부 서브루틴 S의 프레임으로의 참조다. (심화학습에 있는) 그림 8.7은 그 예를 보여준다.

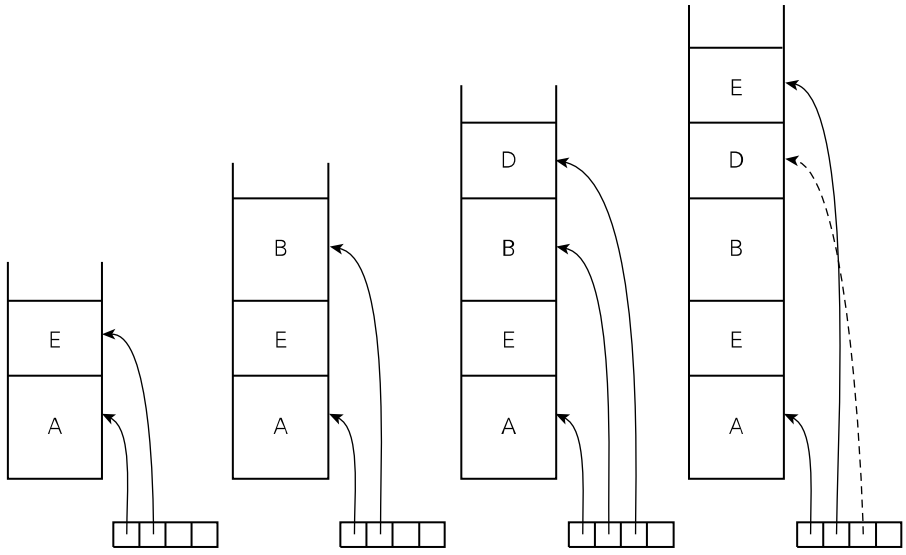

그림 8.7 | 디스플레이를 이용한 비지역 접근. 왼쪽부터 오른쪽으로 스택 환경 설정은 그림 8.1의 어휘적 중첩을 가정하는 연속되는 서브루틴 호출의 내용을 보여준다. 현재 실행 중인 서브루틴의 위에 있는 디스플레이 원소는 사용하지 않는다.

디스플레이를 메모리에 저장한다면 비지역적 객체는 디스플레이 원소를 레지스터로 불러오기 위한 첫 번째 접근과 객체를 불러오기 위한 두 번째 접근, 이렇게 두 번의 메모리 접근을 통해 레지스터로 불러올 수 있다. 다수의 레지스터를 가지고 있는 기계에서 전체 디스플레이를 레지스터에 저장해 메모리 접근 오직 한번으로 오버헤드를 줄이려는 유혹이 있을 것이다. 하지만 이것은 아마도 좋지 않은 생각일 것이다. 디스플레이 원소를 대신 레지스터에 저장할지도 모르는 다른 것(예를 들어 지역 변수)보다 뜸하게 접근하려는 경향이 있다.

설계와 구현

어휘적으로 중첩하기와 디스플레이

디스플레이가 고정된 크기의 배열이기 때문에 비지역 객체에 대한 접근을 구현하기 위해 디스플레이를 사용하는 컴파일러는 일반적으로 서브루틴을 중첩할 수 있는 최대 깊이(디스플레이의 크기)에 제한을 설정한다. 예를 들어 이 제한이 5나 6보다 크다면 어떤 프로그래머도 좀 더 큰 값으로 설정해달라고 원하지 않을 것이다. 디스플레이가 프레임 포인터에 대한 요구를 제거시키지 않음을 유념하라. 지역 변수에 매우 빈번히 접근하기 때문에 어떤 레지스터에 변위 방식 주소 지정을 위해 사용할 수 있는 현재 프레임의 주소를 가지고 있는 것이 중요하다. 비슷한 예로 32비트 주소가 하나의 명령어에 맞추지 못하는 RISC 프로세서에서 가장 빈번히 접근되는 전역 변수에 대한 기준 레지스터를 유지하는 것이 중요하다.

디스플레이를 유지하기

디스플레이의 유지는 정적 체인의 유지보다 약간 더 복잡하다. 그렇지만 훨씬 많이는 아니다. 가장 명백한 접근법은 일반적으로 정적 체인을 유지하고 단순히 프로시저 항목에 디스플레이를 채우는 것과 체인을 따라감에 따라 종료하는 것일 것이다. 하지만 대부분의 경우 (훨씬 더 빠른) 다음 기법이면 충분하다. 어휘 중첩 수준 j의 서브루틴을 호출할 때 피호출자는 j번째 디스플레이 원소의 현재 값을 스택에 저장하고 나서 그 원소를 자신의 (새로 생성된) 프레임 포인터의 사본으로 대체한다. 복귀하기 전에 그것은 이전 원소를 복구한다. 왜 이 기법이 동작하는가? 정적 체인처럼 고려해야 할 다음과 같은 두 가지 경우가 있다.

1. 피호출자는 호출자 내에 (직접적으로) 중첩되어 있다. 이 경우 호출자와 피호출자는 현재 수준까지의 디스플레이 원소 모두를 공유한다. 호출자의 프레임 포인터를 디스플레이로 넣는 것은 단순히 현재 수준을 한 단계 확장시키는 것이다. 이전 값을 저장할 필요가 없는 것은 생각해 볼만하다. 하지만 일반적으로 이를 주장할 방법은 없다. 매우 깊이 중첩되고 매우 깊은 디스플레이의 보전을 기대하고 있는 코드가 호출자 자신을 호출할 수도 있다. 이 경우 이전 디스플레이 원소를 필요로

할 것이다. 똑똑한 컴파일러는 특정 상황에서 이런 저장을 피할 수 있을 것이다.

2. 피호출자는 호출자로부터 $k \geq 0$ 수준 밖에 있는 어휘 중첩 수준 j에 있다. 이 경우 호출자와 피호출자는 $j-1$의 처음부터 끝까지 모든 디스플레이 원소를 공유한다. 수준 j에 호출자의 항목은 피호출자의 것과 다를 것이다. 그래서 피호출자는 자신의 프레임 포인터를 저장하기 전에 그것을 저장해야 한다. 피호출자가 수준 $j+1$에 있는 어떤 루틴을 호출한다면 그 루틴은 디스플레이의 다른 원소를 변경할 것이다. 하지만 그것을 필요로 하기 전에 모든 이전 원소를 복구할 것이다.

호출자가 리프 서브루틴이면 디스플레이를 바꾸지 않은 채 남겨둘 수 있다. 제어가 호출자로 돌아오기 전에 피호출자의 중첩 수준에 일치하는 원소를 사용할 것이다.

클로저

매개변수로 전달하거나 변수를 저장하거나 함수로부터 복귀하는 서브루틴을 참조 환경을 저장한 어떤 종류의 클로저(3.5절)를 통해 호출해야 한다. 정적 체인에 기반한 언어 구현에서 클로저를 (코드 주소, 정적 연결)로 표현할 수 있다. 디스플레이는 그렇게 간단하지는 않다. 전형적인 기법은 모든 서브루틴에 대해 두 "진입 지점", 즉 시작하는 주소를 만드는 것이다. 이것 중 하나는 "일반적인" 호출을 위한 것이고 나머지는 클로저를 통한 호출을 위한 것이다. 클로저를 생성할 때 그것은 대체 진입 지점의 주소를 포함한다. 그 진입 지점에 코드는 해당 디스플레이의 1부터 j까지의 원소를 스택 내에 저장하고(이것을 하기 위해 평소보다 크기가 큰 스택 프레임을 생성해야 할 것이다) 원소를 클로저로부터 취한(혹은 계산한) 값으로 대체한다. 대체 진입은 서브루틴의 주 몸체에 대해 중첩된 호출을 만든다(이는 일반적인 진입에 뒤따라 오는 일반적인 스택 프레임을 만들고 디스플레이를 갱신하는 코드를 즉시 넘어간다). 서브루틴이 복귀하면 대체 진입의 코드로 돌아온다. 이 코드는 실제 호출자로 돌아가기 전에 디스플레이의 이전 값을 복구한다.

디스플레이 기반의 클로저의 경우 좀 더 공간 보존적인 구현은 가능하지만(연습문제 8.5를 확인하라) 더 높은 실행 시간 오버헤더를 가지고 있다.

정적 체인과의 비교

일반적으로 디스플레이를 유지하는 것은 정적 체인을 유지하는 것보다 약간 더 비싸다. 그렇지만 이 비교가 절대적인 것은 아니다. 보통의 경우에 호출된 루틴에 정적 연결을 전달하는 것은 (스택 프레임 내에 적절한 오프셋에 정적 연결을 위치시키기 위해) 호출자 내에 $k \geq 0$ 개의 불러오기 명령어와 그 다음에 하나의 저장 명령어를 필요로 한다. 레지스터가 연결을 저장하기에 충분한다면 리프 루틴에서는 저장을 넘어가도 된다. 서브루틴에서 복귀할 때 정적 체인을 유지하기 위해 어떤 오버헤드도 필요하지 않다. 디스플레이를 사용하면 리프가 아닌 피호출자는 디스플레이 원소를 저장하고 복구하

기 위해 2개의 불러오기와 3개의 저장(프롤로그에서 1+2와 에필로그에서 1+1)을 필요로 한다. 피호출자가 모든 일을 하기 때문에 디스플레이는 정적 체인에 비해 코드 크기를 약간 줄일 지도 모른다. 위에서 언급했듯이 디스플레이는 클로저의 생성과 사용을 복잡하게 한다.

디스플레이의 원래 이점인 외부 유효 범위의 객체로의 접근에 대한 비용의 감소는 오늘날 예전보다 좀 더 불확실해 보인다. 사실 디스플레이는 1970과 1980년대의 CISC 컴파일러에서 인기가 있었지만 최근 컴파일러에서는 그렇지 않다. 대부분의 프로그램은 서브루틴을 2나 3 수준 깊이로 중첩하지 않는다. 그래서 정적 연결이 매우 긴 일은 드물다. 그리고 주변 유효 범위의 변수를 그렇게 빈번히 접근하지 않는 경향이 있다. 자주 접근한다면 공통 하위 수식 최적화가 적절한 프레임으로의 포인터가 어떤 레지스터에 남아있음을 보장할 수 있다.

일부 언어 설계자는 객체지향 프로그래밍(9장의 주제)의 개발이 중첩된 서브루틴의 필요를 제거해왔다고 주장해왔다[Han81]. 다른 사람은 C의 성공이 그런 루틴이 불필요함을 보여주고 있다고 말하기도 한다. 물론 중첩된 서브루틴 없이 정적 체인과 디스플레이 간의 선택은 토론의 여지가 있다.

✓ 확인문제

47. 디스플레이에 기반한 언어 구현에서 어휘적 중첩 수준 k에 있는 객체에 접근하는 방법을 설명하라.

48. 디스플레이를 일반적으로 레지스터 안에 저장하지 않는가?

49. 서브루틴 호출 동안에 디스플레이를 유지하는 방법을 설명하라.

50. 디스플레이를 사용하는 언어 구현에서 클로저를 생성할 때 어떤 특별한 고려가 발생하는가?

51. 디스플레이와 정적 체인 간의 트레이드오프를 요약하라. 디스플레이가 더 빠른 코드를 만드는 프로그램을 설명하라. 정적 체인이 더 빠를 것인 다른 코드도 설명하라.