



사이버 토론 | 데브피아 회원들에게 듣다

‘양날의 검’ 포인터, 지금도 필요할까?

이번 개발자 토론방은 개발자들에게 너무나 친숙한 소재인 ‘포인터’를 토론 주제로 선정했다. 비록 그 존재감이 예전만 못한 게 사실이나 프로그래밍의 개념 구축에 크게 기여해 온 만큼, 현 프로그래밍 환경에서의 실질적인 의미를 되짚어 보기 위해서다. 토론 과정에서 공유되는 개발자들의 다양한 경험과 생각들은 포인터 활용이 여전히 어렵게 느껴지는 초보 프로그래머들에게 많은 도움이 될 것이다.

정리 | 전도영 기자 | mir@imaso.co.kr

Cyber
Talk
www.devpia.com

변수의 메모리 주소를 담는 포인터는 그 자체가 하나의 변수로 여겨지면서 프로그래밍 효율을 높이는 데 큰 역할을 해 왔다. 하지만 이 포인터도 잘못 활용하면 과거 ‘goto 문’이 그랬던 것처럼 만만치 않은 오류를 일으키곤 한다. C 언어가 배우기 어려운 것으로 여겨지고, C 개발자들이 포인터를 ‘양날의 검’에 빚낸 것도 모두 같은 이유인 셈이다.

따라서 이번 개발자 토론방은 너무나 일상적으로 쓰여 왔던 포인터를 재조명하

기 위해 이달의 주제로 선정했다. 프로그래밍 환경이 변화한 탓에 시기적으로 적합하지 않은 토론 주제라는 지적도 있었지만, 오히려 그렇기에 개발자들의 생각이 더 궁금해지기도 했다. 개발 언어와 도구의 변화 속에서 포인터의 의미도 달라졌고, 그 쓰임새와 활용법 역시 미묘하게 변화했을 것이라 기대하는 추측 때문이다.

보다 명확한 토론을 위해 기사는 ‘포인터’의 의미를 지극히 좁은 의미로 한정했다. 직접적으로 메모리 번지를 다루는 번



마소 개발자 토론방은 이렇게 운영됩니다

월간 마이크로소프트웨어와 개발자 천국을 꿈꾸는 IT 포털 데브피아가 궁극한 IT 이슈에 대해 의견을 교환하는 ‘개발자 토론방’ 코너를 운영합니다. 데브피아 게시판과 본지 지면을 통해 진행되는 ‘개발자 토론방’은 현재 IT 산업에 몸담고 있는 수많은 개발자들의 의견을 수렴해 이슈에 대한 이해의 폭을 넓히고, 이를 통해 IT 산업의 흐름을 가능하고자 하는 취지에서 마련됐습니다. 코너 운영은 다음과 같습니다.



데브피아(www.devpia.com)에 마련된 ‘개발자 토론방’ 게시판.

- 1 토론 주제 : 담당 기자와 데브피아 관계자의 협의를 통해 선정한다.
- 2 기간 : 매달 1일부터 20일까지
- 3 토론 진행 : 데브피아가 마련한 ‘개발자 토론방’ 게시판에 온라인 게시물을 작성하는 형태로 진행된다. (온라인 설문조사 병행 실시)
- 5 게시물 작성 : ‘댓글’ 형식으로 특별한 양식없이 작성하므로 누구나 자유롭게 참여할 수 있다. 선정된 의견은 이곳 지면을 통해 소개한다.

온라인 토론은 무엇보다 이용자들의 적극적인 참여가 전제되어야 하는 만큼, 독자 및 데브피아 회원 여러분의 많은 참여를 당부 드립니다.

수, 즉 C와 C++ 등에서 쓰이는 전형적인 포인터만을 고려기로 한 것이다.

과연 포인터의 가치는 어떻게 달라졌을까? 지금부터 프로그래밍 패러다임의

변화와 프로그래밍 도구의 변화, 그리고 메모리 확장을 비롯한 하드웨어 환경의 발전 속에서 포인터에 대한 인식이 어떻게 바뀌었을 지를 살펴본다. 개발자들의

의견을 정리하는 과정에서 제시자의 실제 의도와 다소 차이가 날 수 있음을 밝혀둔다.

- 포인터, 과연 어떤 의미를 지녔을까요?

포인터는 과거 'C 언어의 꽃'에 비유되며 프로그래밍 효율에 있어서 절대적인 가치를 인정받았다. 메모리를 직접 제어할 수 있으므로 포인터만 이용하면 마치 무엇이든 다 구현할 수 있을 것처럼 보였다. 프로그래밍 입문자들 역시 마찬가지. 포인터만 이해하면 C의 모든 것을 배운 듯한 기분이었다. 포인터의 필요를 말하기 전에 새삼스럽지만 포인터의 의미부터 물었다.

김상현(4winners) - '포인터는 양날의 검.' 포인터를 배우고 이용해 오면서 수도 없이 들었던 비유입니다. 맞는 말이에요. 때론 보약이 되고, 때론 독약이 될 수 있는 것이 포인터니까요. 보약을 남용하면 몸에 해가 될 수 있고, 독약도 잘 쓰면 병을 치료하는 것처럼 프로그래밍의 포인터 역시 프로그래밍에 있어서는 종잡을 수 없는 존재입니다.

이영수(getfree74) - 우선 포인터와 참조를 명확히 구분하고 싶어요. 흔히 call by reference라고 하면 참조를 가리키고, 포인터는 call by address의 의미를 가집니다. 그러나 C++에서는 종종 참조와 포인터를 거의 동일한 것으로 취급하므로 주의가 필요합니다. 문법상으로 참조를 포인터를 랩핑(캡슐화)한 개념으로 보기 때문이죠. 오프셋 연산이나 포인터 할당 등에 몇 가지 기능 차이가 있습니다만 실제 처리는 거의 동일하다고 봐야지요. 실제로 C++ 소스 코드를 역어셈블링 해보면 참조와 포인터를 처리하는 코드가 거의 동일한 것을 볼 수 있죠.

김정택(copycd) - 문법적으로 명시적인 포인터를 지원하지 않을 뿐이지 자바, 펄, 파이, C# 등도 내부적으로는 포인터처럼 동작하는 수단을 가지고 있습니다. 만일 그런 것이 없다면 큰 구조체 등을 옮길 때

어려움이 많겠지요. 아울러 함수 파라미터나 변수 등을 넘겨줄 때의 부하도 무시할 수 없습니다.

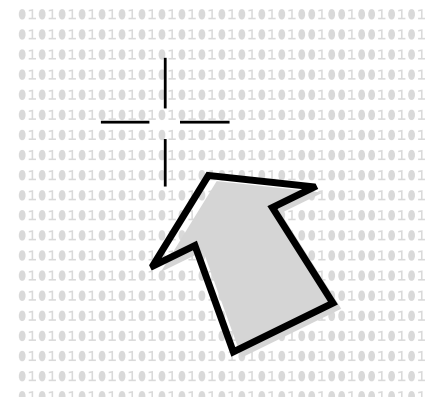
오수환(ooooo) - 학교 다닐 때 포인터에 대한 강의를 여러 차례 들었지만 대체 무슨 말인지 이해하지 못할 때가 많았습니다. 하지만 개발자의 길로 접어들어 직접 프로그래밍을 하다 보니 그 실체를 저절로 알게 되었어요.

이영수(getfree74) - 공감합니다. 포인터를 깊이 파고들면 파고들수록 복잡하게 느껴지는 것은 사실이니까요. 하지만 개념 자체가 절대적으로 어려운 것은 아닙니다. 다만 익숙하지 않을 뿐이지요. 물론 3중, 4중 포인터를 이용하면 정말 어려울 수도 있겠지만, 그것 역시 개념적인 문제라고 생각합니다. 사람은 2차원 구조 이상의 것은 머릿속에 잘 그리지 못합니다.

서광수(chps74) - 어렵게만 느껴지는 포인터를 배울 때는 '왜 포인터를 써야하는가'라는 의문을 누구나 한번쯤 품게 되요. 사실 포인터가 없어도 개발을 못하는 것은 아니니까요. 특히 지금처럼 메모리양과 컴퓨터 성능이 향상돼 데이터 검색이 빠르게 이뤄지는 상황에서는 포인터로 인한 속도 개선은 더 무의미한 것으로 인식될 수 있습니다.

하지만 기계어가 여전히 메모리 번지에 의존하고 있다는 점이 중요합니다. 초기의 시스템 프로그래밍이 모두 어셈블리로 구현됐고, 그 어셈블리를 보다 쉽게 쓰기 위해 C 언어와 포인터가 등장한 것입니다. 하드웨어를 더 쉽게 제어할 수 있는 환경이 마련된 것이죠. 포인터를 사용하는 프로그램이 효율적인 것도 바로 이 때문입니다.

채상혁(serapian) - 버전 업을 멈춘 C/C++이 여전히 실무에 많이 이용되고 있는 것도 바로 포인터 덕분 아닐까요? 실제로 빠른 속도를 요구하는 네트워크 프로그램의 코어 부분은 대부분 C로 작성되어 있습니다.





프로그래밍 환경이 바뀌고 있습니다. 포인터는 지금도 꼭 필요한 도구일까요?

C 언어의 인기가 절정에 달했던 시대에 포인터는 파워풀한 프로그래밍을 위한 필수도구였다. 하지만 프로그래밍 패러다임과 하드웨어 환경 등에 적지 않은 변화가 일어난 지금은 어떨까? 많은 개발자들의 지적대로 포인터의 이용은 더 이상 필요가 아닌 선택이 된 것일까? 실제 토론 과정에서도 '포인터는 구시대의 유물인가'라는 명제를 놓고 활발한 토론이 펼쳐졌다.

허준행(TohnoKanna) - 개인적으로 참조에 의한 호출을 지원하지 않아 포인터를 써야만 하는 C의 방식을 몹시 싫어합니다. 때문에 C#, 자바, 델파이처럼 포인터를 제공하지 않거나 꼭 쓰지 않아도 되는 언어만을 즐겨 쓰죠.

그렇지만 이런 저 역시도 포인터는 모든 언어에 꼭 있어야 한다고 생각합니다. 포인터를 쓰면 더 편리하고 더 효율적인 경우가 있다는 점을 인정하기 때문입니다. 되도록이면 포인터 이용을 줄여 가독성과 안정성을 높이되, 라이브러리 형태의 함수처럼 적재적소에만 포인터를 써서 퍼포먼스에서 이득을 보는 것이 바람직하다는 게 제 생각입니다. 포인터를 사용하면 라이브러리의 한계를 넘을 수 있는 점도 빼놓을 수 없을 테고...

김홍래(hongja) - 포인터가 고급언어로의 진화 과정에서 가장 혁신적인 기법이었음은 잘 알고 있지만, 쓰면 쓸수록 굳이 이걸 이용할 필요를 못 느끼는 게 사실이에요. C와 C++만을 이용할 때는 포인터를 자유자재로 다루는 것에 자부심을 가졌지만, 막상 자바와 C#을 이용해보니 포인터를 쓰지 않는 게 오히려 더 개념적인 프로그래밍에 도움이 됐고 유지 관리도 수월했죠. 언어적 측면이 아니라 사용자의 관점에서 포인터를 달리 보게 되었다고 할까요?

추상 구문의 발전을 고려하면 포인터를 쓰지 않더라도 성능과 알고리즘에는 별 문제가 없어 보입니다. 오히려 포인터가 프로그래밍 입문 과정에 장애가 될지 모르죠.

전현수(wdog777) - 저 역시 포인터는 필요 없다고 생각해요. 개발 환경의 발전에 따라 최근에는 SI 분야에서도 자바 기반의 서버 모듈들이 쏟아지고 있습니다. 그만큼 강력하고, 안정적인 대안이 존재한다는 의미지요.

오수환(ooooo) - 우수한 개발 툴이 대중화되었기 때문인지 포인터의 필요성을 인정하지 않는 의견도 많네요. 하지만 포인터의 개념만큼은 분명히 이해해야 한다고 봅니다. 적어도 call by reference나 call by value와의 차이 정도는 알아야겠죠.

사실 포인터 활용을 잘 하면 메모리 관리에 큰 도움이 되는 것은 틀림없습니다. 배우는 과정에서 이 개념이 과연 얼마나 필요할지를 의심하게 되지만, 프로그래밍을 계속 하다보면 곧 그 가치를 깨닫게 마련이죠.

박세진(gnoses) - 과거 C 언어부터 직접도록 포인터를 이용해왔습니다만, 수년간 델파이와 C#, 자바 등을 경험하면서 포인터는 굳이 필요 없다는 결론을 내렸죠. 실제로 자동 call by reference에 의해 모든 객체는 참조자에 의해 처리되고, 사용이 끝난 메모리는 가비지 콜렉터 및 사용자 소멸 코드에 의해 안전하게 지워집니다. 또한 함수 포인터는 델리게이트(delegate)가, 링크드 리스트는 컬렉션 클래스가 모두 처리해주죠.

물론 바이트 단위 처리나, dll 호출 등에 여전히 메모리 연산이 필요하고, 리눅스나 유닉스 소스 역시 C 문법을 알아야 볼 수 있지만, 이런 코드들은 머지않아 '구시대의 유물'로 사라질 것입니다. 아울러 포

인터 연산이라는 위험한 도구도 이제는 객체 참조와 메모리 풀에 의한 자동 메모리 관리 등에 의해 대체될 수 있어요.

신영진(hacsyj) - 포인터가 과거의 것이라는 의견에는 동의합니다. 하지만 지금과 같은 높은 추상화의 하부 단계가 무엇인지는 한번쯤 생각해봐야 합니다. 누군가가 '구시대의 유물'을 써서 어려운 작업을 해주었기에 그 토대 위에서 지금과 같은 편리한 프로그래밍을 할 수 있는 것이죠. 포인터를 단지 옛것으로 치부하고 학습하지 않는다면, 그런 하부 구현에 점차 의존하게 될 수 밖에 없습니다. 문제가 생겨도 스스로 해결할 힘이 없어지는 거죠. 메모리 어드레싱이라는 패러다임이 변하지 않는 한, 적어도 그것을 배울 만한 가치는 충분합니다. '은고지신'이라는 옛말처럼 말이죠. (웃음)

서광수(chps74) - 저는 구시대의 유물이 아니라 오히려 모든 언어의 근간을 이루는 핵심이라고 봅니다만... 에스컬레이터가 옆에 있는데 걸어서 계단을 오른다고 해서 '구시대 사람'이라고 단정지을 수는 없습니다. 에스컬레이터를 타고 계단을 오른다고 해도 그 끝에 도달하면 다시 걸어야 할지도 모르기 때문이죠. 포인터도 마찬가지입니다. 포인터를 쓰지 않고 할 수 있는 모든 걸 했는데도 여전히 해결해야 할 무엇인가가 남아있다면 어떻게 해야 할까요? 그렇습니다. 그때는 포인터를 써야만 합니다.

이영수(getfree74) - 저 역시 포인터를 구시대의 것으로만 바라보는 시각에는 반

대합니다. 사실 대부분의 경우 포인터는 개발 언어에서 감춰져 있을 뿐이지, 완전히 사라진 것은 아니기 때문이죠. 가비지 컬렉션을 무언가가 대신 해준다고 해서 가비지 컬렉션 자체가 사라지는는 않죠. List와 Tree, Graph, Heap 등의 자료구조는 이미 누군가가 훌륭히 구현해 놓아 이용자가 그것을 새롭게 만드는 경우는 드물습니다. 다만 그 가운데 어떤 것을 가져다 쓸 지를 판단하기 위해 그 자료구조에 대한 학습은 필요하겠죠. 포인터의 경우가 이와 같지 않을까요?

김상현(4winners) - 많은 이들이 어려워하는 C#의 델리게이트는 C의 평선 포인터에서 파생된 것입니다. 실제로 C 언어의 평선 포인터는 C#의 델리게이트로 매핑되어 프로그래밍 되곤 합니다. 이처럼 포인터는 많은 곳에서 유효하게 쓰이고 있어요. 어셈블리에 주소 값이 없으면 의미가 없듯이 포인터가 없는 C 계열 언어는 생각할 수 없습니다. C를 처음 배울 때는 빠른 처리 속도가 요구되는 주요 알고리즘에 어셈블리를 쓴다고 들었습니다만,

요즘에는 이런 주요 알고리즘에 C가 이용되지요. 아마도 포인터 때문일 겁니다.

오수환(ooooo) - 일반적으로 웹 프로그래밍 분야에서는 포인터를 이용하지 않고도 우회적으로 코딩할 수 있는 여지가 큰 반면에, 일반 애플리케이션 개발에서는 포인터를 이용하지 않으면 훨씬 많은 시간과 노력이 뒤따르는 경우가 많습니다. 따라서 포인터의 실질적 가치는 웹 개발 환경이나 애플리케이션 개발 환경이나에 따라 적지 않은 차이를 나타낼 수밖에 없죠.

한중현(hanbell) - 틀림없는 것은 포인터가 부지불식간에 여전히 많은 언어에서 이용되고 있다는 점입니다. 참조자 역시 포인터의 다른 형태로 볼 수 있고, 객체 지향의 측면에서 보면 this, super 모두 포인터와 같습니다. 물론 구조를 복잡하게 만들어 가독성을 떨어뜨리는 단점이 있지만, 이를 두려워 해 굳이 쓰지 않을 이유도 없습니다. 필요할 때는 써야한다는 의미겠죠.

척 까다로워지겠죠.

서광수(chps74) - 컴퓨터의 성능이 발전하면서 코드의 효율성보다는 가독성과 디버깅 효율을 중시하는 게 사실입니다. 유지보수나 리팩토링에 더 이로운 방식을 선택하는 셈이죠.

그렇다고 해서 포인터를 써야 할 곳에 포인터를 쓰지 말아야 하는 것은 아닙니다. 오히려 포인터 이용을 너무 절제하다 보면 스택 관리 등에 문제가 생길 수 있습니다. 무엇이든 과하거나 모자라면 해가 될 수 있죠. 따라서 효율을 고려해 적절히 포인터를 사용하는 '프로그래밍 미학'을 실천해야 할 것입니다.

서영덕(salpa) - 같은 생각입니다. 사실 포인터가 위험하다는 생각은 그만큼 포인터에 익숙하지 못함을 의미하죠. 메모리 침범? 물론 발생할 수 있습니다. 그러나 그건 어디까지나 변수의 잘못된 사용에서 비롯된 문법적 에러일 뿐이지 포인터 자체의 문제는 아닙니다.

사실 아직까지도 어셈블리가 프로그래밍 곳곳에 양념처럼 스며들어 성능 향상에 탁월한 기여를 하고 있는 마당에, 포인터의 효용 여부를 따지는 것은 너무나도 성급해 보일뿐입니다.

정말 중요한 것은 개발 언어가 아니고 프로그래밍의 목적이예요. 언어는 단지 수단에 불과합니다. 따라서 하드웨어나 운영체제의 변화, 개발 도구의 발전 등은 포인터의 이용 여부를 판단하는 데 아무런 근거가 되지 못합니다. 목적에 맞도록 적절한 언어를 선택해 개발하고, 그 과정에서 포인터가 필요하다고 생각되면 이용하면 그만이니깐요. 당연한 말이었지만 포인터를 써야한다고 판단되었을 때 정말 제대로 쓰는 게 더 중요한 이슈가 되겠죠. +

그렇다면 포인터를 어떻게 받아들이고 써야할까요?

그것이 바람직하든 바람직하지 않든 간에 포인터의 필요성을 인정하는 의견이 아직까지는 조금 더 우세한 것으로 나타났습니다. 어떤 개발 언어라도 그 밑바탕에는 포인터의 개념이 자리 잡고 있다는 것이 그 필요성을 인정한 결정적인 근거다. 그렇다면 개발자들은 앞으로 포인터를 어떻게 이해하고 받아들여야 할까? 마지막 질문을 통해 포인터에 대한 생각을 정리했다.

채상혁(serapian) - 컴퓨터의 시스템 설계를 학습하다 보면 포인터에 대한 이해가 더 빨라지는 게 사실입니다. 예를 들어 동적 할당 등을 수행한 후 실제 디버깅하면서 어셈블리 코드를 살펴보면 포인터와 메모리 주소와의 관계 등이 훨씬 쉽게 파악됩니다. 비록 포인터가 쉽지 않은 개념입니다만, 소프트웨어가 하드웨어에 종속적일 수밖에 없는 상황을 고려하면 하드

웨어에 조금 더 접근할 수 있는 통로로 포인터가 유용하게 쓰일 수 있겠죠.

허준행(TohnoKanna) - 라이브러리 가운데 C로 만들어진 것이 가장 많다는 점을 감안하면 포인터 활용의 중요성은 더 커집니다. C로 작성된 유용한 소스를 가져와 이용할 때 만일 개발 언어가 포인터를 지원하지 않는다면 그 변환 과정이 무