역참조 연산자 *는 포인터 앞에 붙입니다. 다음과 같이 numPtr 앞에 *를 붙이면 numPtr에 저장된 메모리 주소로 가서 값을 가져옵니다. 여기서는 numPtr이 num1의 메모리 주소를 저장하고 있으므로 num1의 값인 10이 출력됩니다.

```
printf("%d\n", *numPtr);    // 10: 역참조 연산자로 num1의 메모리 주소에 접근하여 값을 가져옴
```

즉, 포인터는 변수의 주소만 가리키며 역참조는 주소에 접근하여 값을 가져옵니다.

그림 34-6 역참조 연산자를 사용하여 메모리 주소의 값을 가져옴

포인터는 변수의 주소만 카리킴


역참조는 주소에 접근하여 값을 가져옴


참고 | **포인터 선언과 역참조?**

포인터를 선언할 때도 *를 사용하고 역참조를 할 때도 *를 사용합니다. 같은 * 기호를 사용해서 헷갈리기 쉽지만 선언과 사용을 구분해서 생각하면 됩니다. 즉, 포인터를 선언할 때 *는 **"이 변수가 포인터다"**라고 알려주는 역할이고, 포인터에 사용할 때 *는 **"포인터의 메모리 주소를 역참조하겠다"**라는 뜻입니다.

```
int *numPtr;              // 포인터. 포인터를 선언할 때 *
printf("%d\n", *numPtr);  // 역참조. 포인터에 사용할 때 *
```

이번에는 포인터 변수에 역참조 연산자를 사용한 뒤 값을 저장(할당)해보겠습니다.

- *포인터 = 값;

dereference_assign.c

```c
#include <stdio.h>

int main()
{
    int *numPtr;      // 포인터 변수 선언
    int num1 = 10;    // 정수형 변수를 선언하고 10 저장

    numPtr = &num1;   // num1의 메모리 주소를 포인터 변수에 저장

    *numPtr = 20;     // 역참조 연산자로 메모리 주소에 접근하여 20을 저장

    printf("%d\n", *numPtr);    // 20: 역참조 연산자로 메모리 주소에 접근하여 값을 가져옴
    printf("%d\n", num1);       // 20: 실제 num1의 값도 바뀜

    return 0;
}
```

실행 결과

```
20
20
```

역참조 연산자는 값을 가져올 수도 있고 값을 저장할 수도 있습니다. 여기서는 `*numPtr = 20;`과 같이 `numPtr`에 저장된 메모리 주소에 접근하여 20을 저장했습니다. 따라서 `printf`로 `*numPtr`을 출력해보면 20이 나옵니다.

또 한가지 중요한 점은 `*numPtr = 20;`으로 20을 저장한 뒤 `printf`로 변수 `num1`의 값을 출력해보면 20이 나온다는 것입니다. 왜냐하면 `numPtr`에는 `num1`의 메모리 주소가 저장되어 있으므로 역참조 연산자로 값을 저장하면 결국 `num1`에 저장하게 됩니다.