09\_포인터

학습목표

포인터의 개념을 이해한다.

포인터 선언 및 초기화 과정을 이해한다.

포인터의 연산의 특수성을 이해한다.

포인터와 배열의 관계를 이해한다.

포인터를 이용한 참조에 의한 호출을 이해한다.

1. 포인터란?

포인터(pointer)는 메모리의 주소를 가지고 있는 변수이다. 포인터는 변수가 저장되는 주소와 깊은 관계가 있다.

* 1. 변수는 컴퓨터 메모리에 만들어진다.

변수는 어디에 만들어지는 것일까? 변수는 자료를 저장하는 역할을 하므로 메모리(memory)에 만들어진다. 메모리는 바이트로 구성되고 각 바이트마다 고유한 주소를 가지고 있다. 프로그램에서 변수를 만들면 이들 변수를 컴파일러가 메모리 공간에 배치한다.

* 1. 변수의 주소

C언어에는 변수의 주소를 계산하는 연산자 &가 있다. 주소 연산자 & 는 변수의 이름을 받아서 변수의 주소를 반환한다. 메모리 공간에 변수를 배치하는 것은 컴파일러의 권한이라서 우리가 마음대로 할 수 없다.

* 1. 포인터

포인터(pointer)는 가리킨다는 뜻의 동사 point에 er를 붙인 것이다. 따라서 가리키는 것이라는 뜻이다. 포인터는 변수의 주소를 가지고 있는 변수이다. 포인터가 저장하고 있는 것은 값이 아니라 변수의 주소이다.

Ex) int \*p;

P는 정수는 가리키는 포인터이다. 포인터도 변수이다. 포인터는 사용하기 전에 반드시 초기화해야 한다. 포인터에는 변수의 주소가 저장되어야 하므로 &연산자를 이용하여 변수의 주소를 계산하여 포인터에 대입하여 주면 된다.

Ex) int numer = 10;

Int \*p;

P = &number;

변수 number의 주소가 4이었다고 가정하면 포인터 p에 4가 저장된 것이다. 포인터는 지정된 자료형의 변수만을 가리킬 수 있다.

* 1. 간접 참조 연산자\*

포인터가 단순히 메모리의 주소만 저장할 수 있는 것이라면 별로 유용하지 않을 것이다. 포인터가 유용한 이유는 포인터를 통하여 포인터가 가리키는 위치의 값을 읽어오거나 변경할 수 있기 때문이다. 포인터 p가 가리키는 위치에 저장된 값을 가져오려면 p앞에 \*기호를 붙여서 \*p하면 된다. 이것을 포인터를 통하여 간접 참조(dereferencing, indirection)한다고 한다.

\*연산자는 단항 연산자로서 괄호 다음으로 높은 우선순위를 가진다.

포인터 연산자 1. 주소연산자(&) 2. 간접 참조 연산자(\*)

1. 포인터 연산
   1. 포인터에 대해서는 덧셈과 뺄셈 연산만 가능하다. 포인터 변수에 대한 연산은 일반적인 변수에 대한 연산과는 조금 다르다. 포인터에 증가연산인 ++를 적용하였을 경우, 증가되는 값은 포인터가 가리키는 객체의 크기이다. 따라서 char형 포인터를 증가시키면 char형의 크기인 1바이트만큼 증가한다. Int형 포인터를 증가시키면 int형의 크기인 4바이트만큼 증가한다. –연산자를 사용하여 감소시킬 때도 마찬가지이다.
2. 함수와 포인터

포인터는 어디에 사용되는 것일까? 다른 사람에게 넘겨주어야 하는 정보가 상당히 방대하다고 하자. 이런 경우에는 전체를 복사해서 넘겨주는 것보다는 페이지 수만 알려주는 편이 간결할 수 있다. 데이터를 전부 복사해서 함수로 넘기는 것보다 데이터가 있는 위치를 포인터로 알려주는 편이 효율적이다.

일반적으로 프로그래밍 언어에서 함수가 외부로부터 데이터를 받는 방법에는 2가지가 있다.

1. 값에 의한 호출(call-by-value): 함수가 호출될 때 복사본이 함수로 전달되면 값에 의한 호출이다. 함수 안에서 매개 변수를 변경하여도 원보에는 영향을 주지 않는다.
2. 참조에 의한 호출(call-by-reference): 만약 함수가 호출될 때 원본을 함수로 전달하는 방법이다. 함수 안에서 매개 변수를 변경하면 원본 변수가 변경된다.

C언어에서는 기본적으로 “값에 의한 호출”만 가능하다. 즉 함수 안에서 매개 변수를 변경하더라도 원본 변수는 변경되지 않는다. 하지만 포인터를 함수에 전달하면 “참조에 의한 호출”을 흉내 낼 수 있다.

값에 의한 호출 방법은 버그를 막을 수 있는 좋은 정책이다. 하지만 경우에 따라서는 원본을 그대로 보내는 것이 필요한 경우도 있다. 포인터를 이용한 “참조에 의한 호출” 흉내내기를 해보자.

1. 포인터 사용시 주의할 점
   1. 초기화하지 않고 사용하기

포인터는 강력한 도구이지만 제대로 된 이해 없이 사용할 경우 프로그램에서 오류를 일으키는 원천이기도 하다. 만약 포인터가 선언만 되고 초기화되지 않았다면 포인터는 임의의 주소를 가리키게 된다. 따라서 이런 상태에서 포인터를 이용하여 메모리의 내용을 변경한다면 문제가 발생할 수 있다. 예를 들어 포인터 p를 초기화 시키지 않고 값 100를 대입하는 코드는 매우 위험한 코드이다.

Int \*p;

\*p = 100; << 매우 위험한 코드

만약 포인터 p가 중요한 부분을 가리키고 있었다면 그 값이 지워지고 100으로 덮어 쓰여져 치명적인 오류가 발생할 수 있다.

포인터가 아무것도 가리키고 있지 않을 때는 NULL(0)로 설정하는 것이 바람직하다.

1. 배열과 포인터

배열과 포인터는 아주 밀접한 관계를 가지고 있다. 왜냐하면 배열 이름 그 자체가 포인터이기 때문이다. 배열 이름은 첫 번째 배열 원소의 주소와 같다. 배열 요소들은 메모리에서 연속된 공간을 차지하고 있다. int형 배열이라고 가정하면 각 요소들이 차지하는 공간은 4바이트이다. 놀라운 사실은 배열의 이름을 정수 형식으로 출력하면 배열의 첫 번째 요소의 주소와 같다는 사실이다.

포인터를 배열처럼 사용

포인터도 배열 이름처럼 간주될 수 있고 배열과 똑같이 사용할 수 있다. 이것은 상당히 편리한 기능이다. 이것을 어디에 사용할까? 대용량의 데이터가 필요한 함수가 있다고 하자. 데이터를 복사해서 주면 너무 많은 시간이 걸린다. 이런 경우에는 데이터가 있는 위치를 알려주고 필요한 부분만을 사용하도록 하면 좋을 것이다.

개인적인 정리

Int \*p; 포인터를 선언하면 p에는 주소 값이 \*p에는 그 위치의 값이 저장된다.