C프로그래밍 실습

**#10 Pointers**

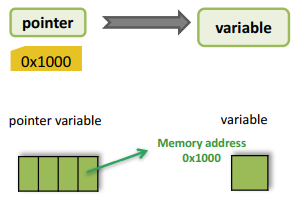
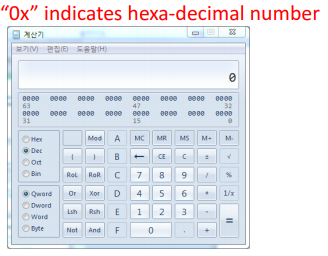
* Pointers

+ 모든 변수는 메모리 공간에 위치된다. 그리고 시작 메모리 주소로 식별된다.

+ pointer: int, char, … 와 같은 다른 변수들의 메모리 주소를 가진 변수이다.

(포인터의 메모리 주소를 포함하기도 한다. => 메모리 주소의 메모리 주소)

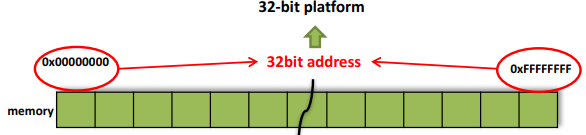
0x의 형태 즉, 16진수의 형태로 저장된다.

* Memory address

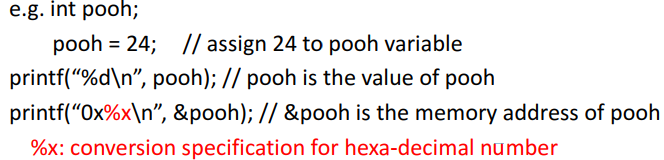
메모리 주소의 크기는 플랫폼에 따라서 다르다.

일반적으로는 32-bit (4 byte)



* Finding Addresses: The & Operator

& 연산자 : 저장된 변수가 있는 메모리 주소를 반환한다.



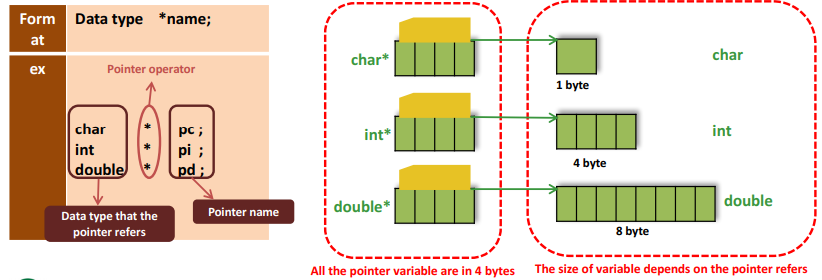


* Declaring Pointers

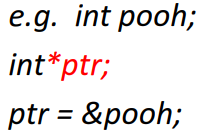
\* 연산자를 data type과 포인터 변수 이름사이에 넣는다.

* Data type \*name;

모든 포인터 변수는 4byte (주소크기)이다. 포인터가 지칭하는 데이터 형에 관계없이



포인터는 다른 변수의 메모리 주소를 저장하는 데 사용되는 변수이다.



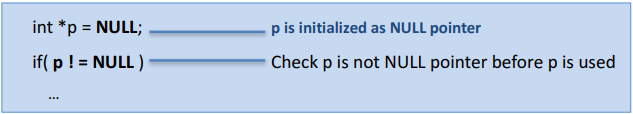
pooh변수의 메모리 주소를 ptr 포인터 변수에 할당한다.

* NULL Pointer

포인터는 0혹은 NULL로 initialized된다.

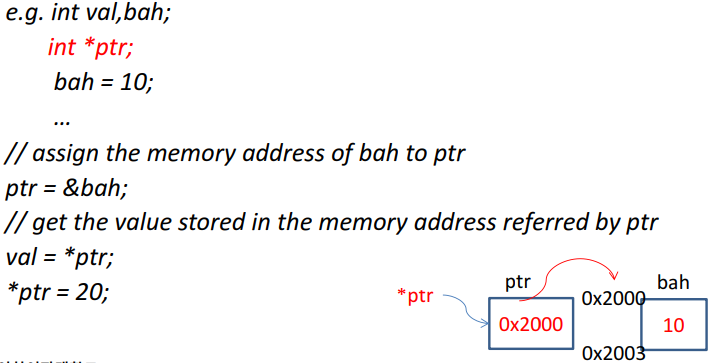
일반적으로 NULL pointer는 메모리 주소나 다른 변수를 참조하지 않음을

나타내기 위해 사용된다.



* The Indirection (Dereferencing) Operation: \* (역참조 연산자 \*)

포인터 변수에 의해 참조된 메모리 주소에 접근(read or write)했을 때의 값



* Usage of Pointers

+ scope밖에 있는 변수를 수정할 수 있다.

또한 함수에서 2개이상의 값을 반환 할 수 있다.

+ 복잡한 인수를 배열과 같은 함수에 전달한다.

* scope밖에 있는 변수 수정하기

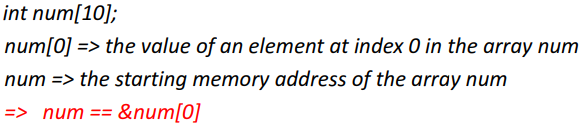
+ 함수에 데이터 변수의 값 대신 메모리 주소(포인터 값) 전달

+ 포인터의 &와 \* 연산자를 사용해서 변수를 읽거나 쓰기

* declaration: void function (data variable \*name);
* call: function (&name);
* Pointers and Arrays

+ 만약 [index]없이 배열 이름을 사용해야하는 경우

+ 배열 자체의 이름이 배열(또는 포인터)의 시작 메모리 주소가 된다.



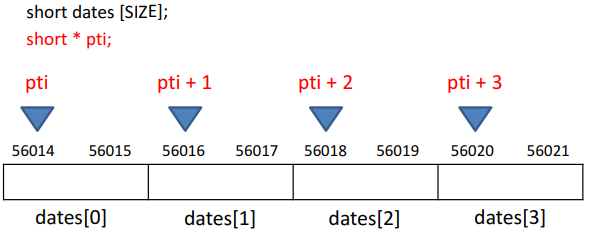
+ 두 값을 포인터 변수에 할당 할 수 있다.

+ 포인터 변수의 더하기 빼기 연산자

- 메모리 주소는 저장 단위(데이터 형의 크기)에 의해 연산된다.

- 주소는 다음 주소로 증가 (pointer +1) 혹은 감소 (pointer -1)한다.

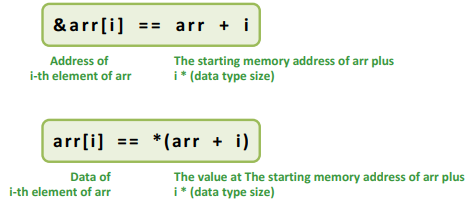
+ 저장 단위는 포인터 변수의 type에 의해 결정된다.



+ arr은 배열의 이름이다. 그리고 그것은 arr 배열의 시작 메모리 주소이다.

그러므로, arr 배열에 대한 포인터로 사용 될 수 있다.

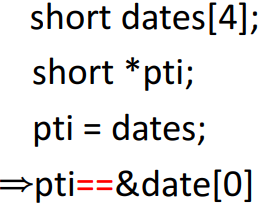
- arr[i]는 \*(arr+i)와 같다.



* Functions, Arrays, and Pointers

+ 배열을 함수에 parameter로 전달하는 방법은?

= 배열에 대한 포인터를 사용한다. (= 배열의 이름을 사용한다.)

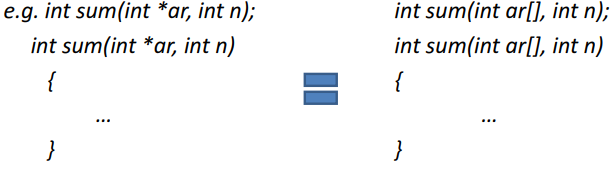






* declaration: void function (data variable \*name)
* function안에서 배열을 사용할 때는 name[i]
* call: function (name);
* Declaring Array Parameters

포인터 대신 parameter로 배열을 전달할 때는 빈 대괄호를 사용 할 수 있다.



* Function Prototype with Array

int sum(int \*ar, int n) == int sum(int ar[], int n);

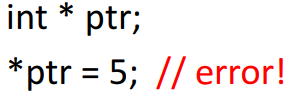
== int sum(int \*, int n) == int sum(int [], int n);

* Pointer and Memory Size

sizeof() 함수는 해당하는 배열 사이즈 X 데이터형의 byte를 계산한다.

포인터변수에 sizeof를 사용하면 4byte가 나온다.

* Dereferencing and Uninitialized Pointer



ptr이 가리키는 위치에 5를 저장한다.

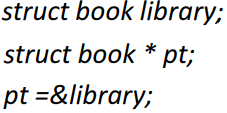
ptr이 initialized되지 않았으며 임의의 값이 있다 => 5가 어디에 놓일 지에 대해 모른다.

* Pointers to Structures

+ struct는 struct에 대한 포인터를 사용하여 함수에 인수로 전달 할 수 있다.

+ Declaring and Initializing a structure pointer

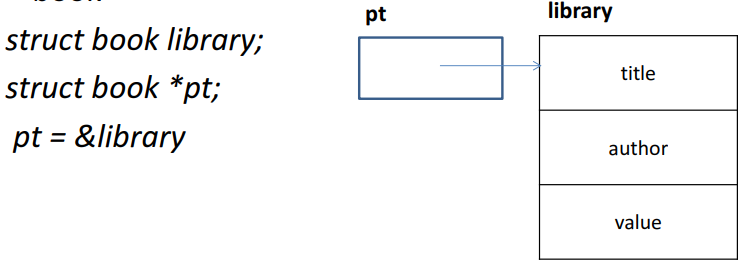
=일반 데이터 유형에 대한 포인터 선언과 동일하다.



+ struct book \*pt;

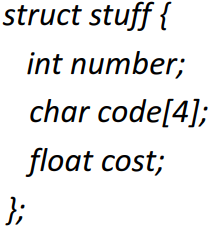
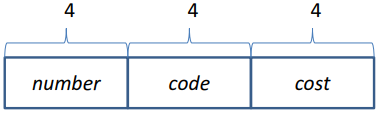
= 새로운 structure를 만들지 않는다.

= 그러나 struct book 데이터 형을 가진 구조체를 가리킨다.



* Memory allocation for a structure

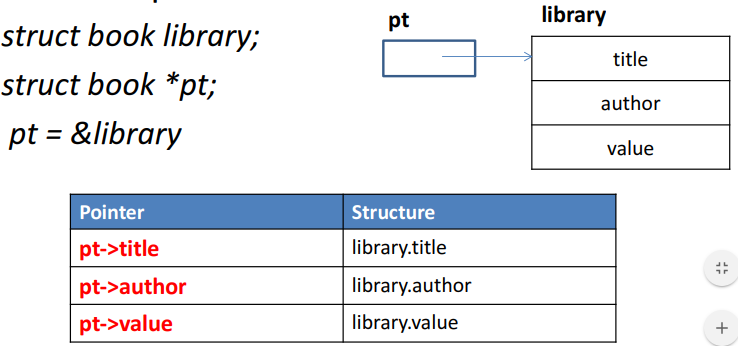
+ 구조체 내의 각 변수는 순서대로 할당된다.



* Member Access by Pointer

구조체 멤버에 접근하는 것과는 다르다!

“->” 연산자를 사용한다.



* Using the Structure Pointer for Passing Structure Variable to Function

+ 구조체 멤버 전달

- 구조체 멤버가 단일 값을 가진 데이터 유형인 경우에

- 각 데이터 변수를 구조체 변수에 개별적으로 전달한다.

ex) library.title, library.author, library.value

+ 구조체를 포인터로 전달하기

- 구조체의 주소 (포인터) 전달

배열을 함수에 전달하는 것과 비슷하다.

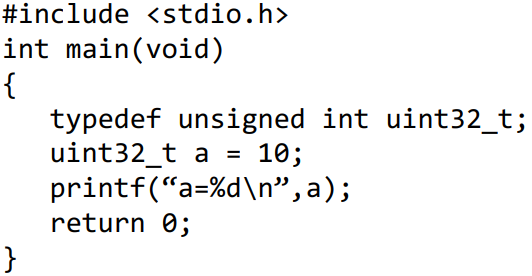
+ 구조체를 값으로 전달하기

- 구조체 값 전달

함수에 값을 전달하는 것과 비슷하다

=함수에서 멤버를 변경해도 원래 구조체 멤버에는 영향을 주지 않는다.

* Typedef: define Custom data type
* typedef [original data type] [new data type];



* Typedef on Structure

typedef는 보통 구조체에서 사용되며 구조체 이름을 생략 할 수 있다.

* type def struct {

float coef;

float exp;

} polytype; 익명의 구조체를 정의했다 coef와 exp를 멤버로 갖고있는