7 포인터 I

7-1 포인터 들어가기: 주소의 개념과 포인터의 정의

7-1-1 주소의 개념과 주소 연산자 &

7-1-2 포인터의 정의

7-2 포인터 변수의 선언과 사용

7-2-1 포인터 변수의 선언

7-2-2 포인터 변수의 사용

7-2-3 포인터 변수 사용 시 유의 사항

7-3 다차원 포인터

7-4 동적 할당

7-4-1 동적 할당의 필요성: 메모리를 자유롭게 활용하자

7-4-2 동적 할당과 해제

7-4-3 동적 할당에서의 유의 사항

@ 더 알아보기: void 포인터

7-1 포인터 들어가기: 주소의 개념과 포인터의 정의

지금까지 연산자, 제어문, 구조체 등 여러 C 프로그래밍 문법들을 배워 보았다. 이번 장에서는 C언어의 꽃이라 불리우는 ‘포인터’의 개념에 대해 배울 것이다. 이 개념을 배우고 잘 익히면 더욱 수준 높은 프로그램을 작성할 수 있다. 일상생활에서의 예를 들어 주소 값과 포인터의 개념에 대해 알아보도록 하자. 독자들은 2장 변수 파트에서 변수를 ‘재료를 담는 상자’라고 표현했던 것을 기억할 것이다. 상자를 가지고 있다고 해서 항상 사용할 수 있는 것은 아니다. 상자가 어디 있는지, 어디에 놓았는지 알아야 된다. 마찬가지로 필요한 재료를 상자에 담아 놓았다고 끝나는 것이 아니다. 분명히 상자에 넣어놨는데, 그 상자가 어디 있는지를 모른다면, 결국 그 재료를 사용할 수 없다. 이와 같이 상자의 위치가 바로 변수의 주소 값이라고 말할 수 있다.

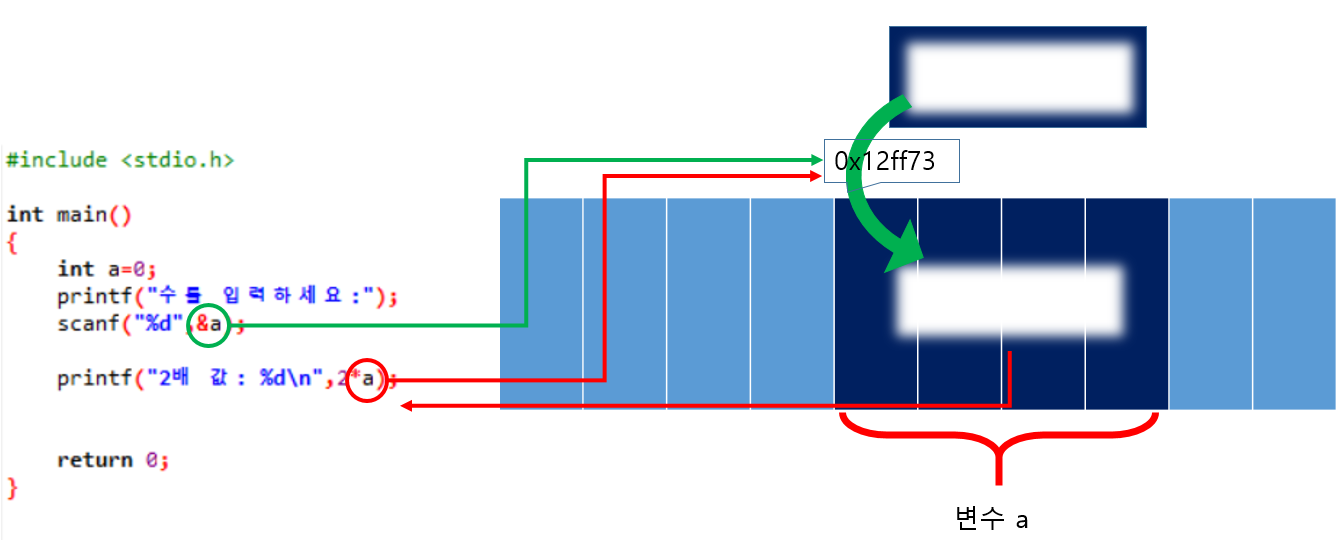
이제 우리가 배울 ‘포인터’라는 개념은 상자의 위치를 적어둔 표지판, 즉 다른 변수의 주소를 저장하는 변수이다. 예를 들자면 빨간 상자는 안방 화장대 밑에 있고 노란 상자는 냉장고 위에 있다는 것처럼, int형 변수 a의 위치, double형 변수 b의 위치를 나타내는 것이 포인터이다. 변수의 위치라는 표현이 잘 와 닿지 않을 수도 있다. 컴퓨터 메인보드 뚜껑을 열어 보면 메모리 혹은 RAM이라고 부르는 반도체가 있다. 독자 여러분이 컴퓨터를 살 때의 기억을 떠올려보자. 4GB 혹은 8GB 램이라고 홍보하던 것을 기억할 것이다. 이 램이라는 저장공간 내부에서 변수가 어디에 있는지를 나타내는 것이 변수의 주소 값을 뜻한다. 이렇게 포인터를 이용하면 각 변수의 이름이나 저장하고 있는 값이 아닌 그 변수들의 위치만 기억함으로써 변수를 더욱 자유롭게 이용할 수 있다.



(그림. 방 그림 : 책상 밑에 있는 상자, 옷장 옆에 있는 상자 등등)

7-1-1 주소의 개념과 주소 연산자 &

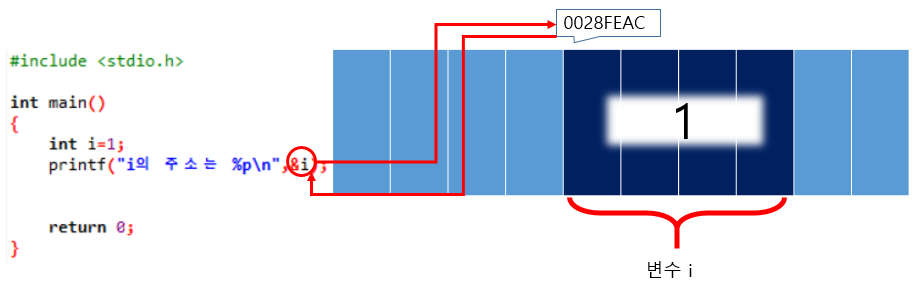
포인터에 대해 배우려면 주소에 대한 개념이 바로 서야 한다. 기억이 날 지 모르겠지만, 우린 1장에서 printf와 scanf에 대해 잠깐 배웠었다. 그 중 scanf를 사용할 때, &(엠퍼센트, ampersand) 라는 연산자를 사용했었다. 다음 코드를 보면서 그 때의 기억을 떠올려 보자.



‘&’ 연산자는 해당 변수의 주소를 알려주는 연산자이다. 위의 코드 내용을 분석해보면, 상자 a에다가 값을 받고 그 두 배의 값을 출력하는 코드이다. 그런데 scanf에서 그냥 a가 아니라 &a인 이유는, 컴퓨터가 프로그래머의 소스 코드를 해독할 때(컴파일 할 때), 모든 변수의 값은 주소로 접근을 한다. 그 의미는 각각의 상자 이름으로 접근 하는 것이 아니라, 그 상자들의 주소를 통해 접근을 하게 된다. 일상에서도 편지를 배달할 때, 철수네, 영희네 라는 표현을 쓰는 대신에 광성로 37-2 번지 와 같은 구체적인 주소를 사용한다. 대한민국에는 수많은 철수와 영희가 있고, 철수네, 영희네라고 하면 우편집배원은 편지를 전달해야 할 정확한 주소가 어디인지 모르기 때문에 제대로 편지를 전달해 줄 수 없다. 이와 마찬가지로 컴퓨터에서도 변수의 이름이 아닌 변수의 주소로 접근해주어야 한다.



그렇다면, 우리가 이용한 상자의 위치, 즉 변수의 주소를 구하는 프로그램을 코딩해보자.





독자들은 아마도 처음 보는 %p에 대해 당황할 수도 있고, 왜 16진수의 숫자가 출력되었는지 궁금할 수도 있다. 이를 설명하기 위해서는 조금의 부연 설명이 필요한데, 아래의 내용을 꼭 정독해주기를 바란다.

대한민국에서 집 주소는 2016년 현재 도로명 주소라는 표기법을 이용하고 있다. 위치 또는 주소를 나타내는 방법은 매우 다양하지만, 우리는 ‘도로명 주소’라는 특정한 형식에 맞추어 주소를 나타낸다. 마찬가지로, 컴퓨터에서도 이런 특정한 형식이 존재한다.

컴퓨터는 기본적으로 0과 1로 이루어진 2진수 체제를 이용하고 있다. 이는 반도체에서 전기 신호를 흘려 보내느냐, 전기 신호를 보내지 않느냐를 1과 0으로 구분하는 것을 기본으로 정보를 저장하기 때문이다. 그러나 0과 1로만 구분을 하게 되면 컴퓨터를 설계하는 사람이나 프로그래머는 너무나 괴로운 상황에 빠진다. 그래서 이 숫자들을 몇 가지 묶음 단위로 이용하는 방법을 고안하게 되었는데, ㄱ) - 2진수를 3개씩 묶어서 이용하는 방법과, ㄴ) - 2진수를 4개씩 묶어서 이용하는 방법 두 가지가 있다. 두 특징을 표를 통해 알아보도록 하자.



ㄴ)에서도 컴퓨터 운영체제에서의 차이가 있다. x86 프로세서, x64 프로세서 라는 것을 들어보았을 것이다. 두 프로세서의 차이점은 다음과 같다.



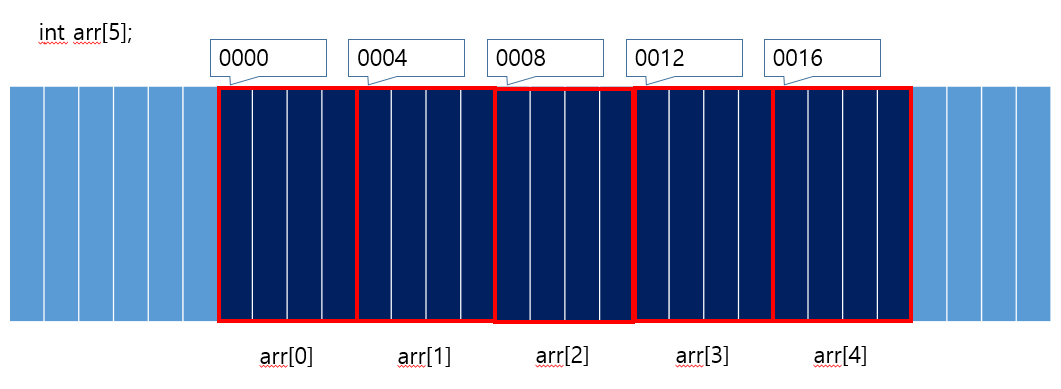
요즘에는 x64 기반의 cpu나 운영체제를 많이 이용하지만, 프로그램의 기본을 설명하는 이 책에서는 좀 더 classic한 x86 기반을 기준으로 설명한다. 따라서, 32bit 기준으로는 모든 변수들은 16진수 8자리의 주소 값을 가지게 된다. 그리고 우리가 앞에서 배운 %x는 16진수를 출력하는 것인데, %x로 변수의 주소를 출력하게 되면 8자리 혹은 16자리가 모두 출력되지 않는다. 그래서 7장 포인터 단원에서부터는 주소 값을 제대로 출력하는 %p를 이용하기로 한다.

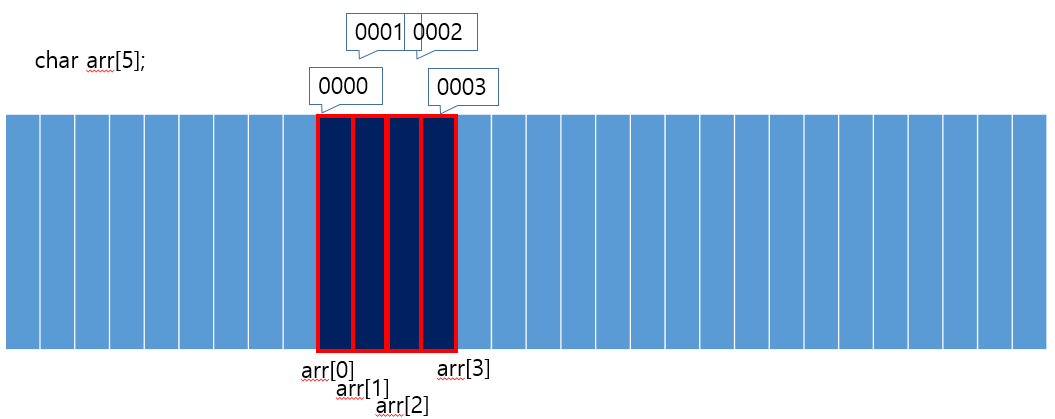
조금 길었지만 포인터는 기본적으로 주소를 저장하는 변수이고, 컴퓨터에서 이 주소체계가 어떻게 운용되는지를 위의 설명을 통해 알아보았다.

그렇다면 변수의 일종인 배열의 경우에는 어떨까?

배열의 메모리 구조를 되새겨보면, 아래 그림과 같은 것을 알 수 있다.

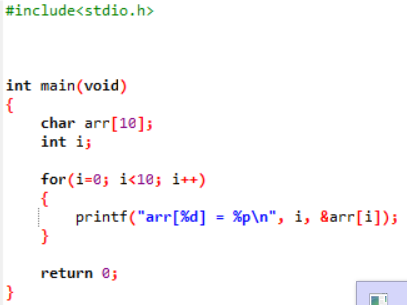
(그림. 배열의 메모리 구조)

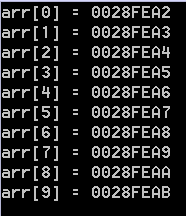




**배열의 이름은 배열의 주소를 나타낸다**는 것을 문자열과 배열 단원 ‘더 알아보기’에서 봤을 것이다. 이는 매우 중요한 사실이고, 따라서 배열의 경우에는 연산자 &를 붙이지 않아도 배열의 주소를 접근할 수 있다. 먼저, 아래 코드로 각각의 배열 요소들의 주소가 어떻게 되어있는지 알아보자.

(코드. int형 char 등의 배열로 for문 이용해서 주소 출력하는 법)

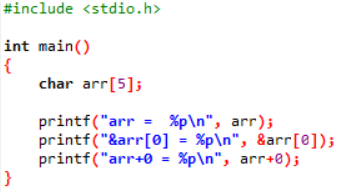


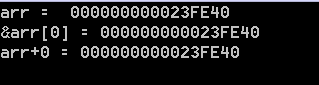


결과를 확인해보면, 각각의 데이터 형이 차지하는 크기만큼씩 주소 값(16진수의 수)이 증가하는 것을 알 수 있다. int형은 4바이트씩, char형은 1바이트씩 증가한다. 변수들이 연속적으로 모여있는 것이 배열이니 주소도 연결되어있는 것은 당연하다.

위의 설명에서 배열에는 &를 붙이지 않아도 된다고 했지만, 여기에서는 &arr[i]와 같은 방식으로 &를 붙였다. 이에 의문을 품는 독자들도 있을 것이다. 이에 대해서는 이번 7장의 나머지를 학습하면서 차차 알게 될 것이다.

참고로, 배열을 주소를 확인할 때 다른 방법들을 이용해서 확인할 수도 있다. 일단은 이런 것이 있다는 것만 알아놓고, 자세한 설명은 8장에서 다루도록 하자.



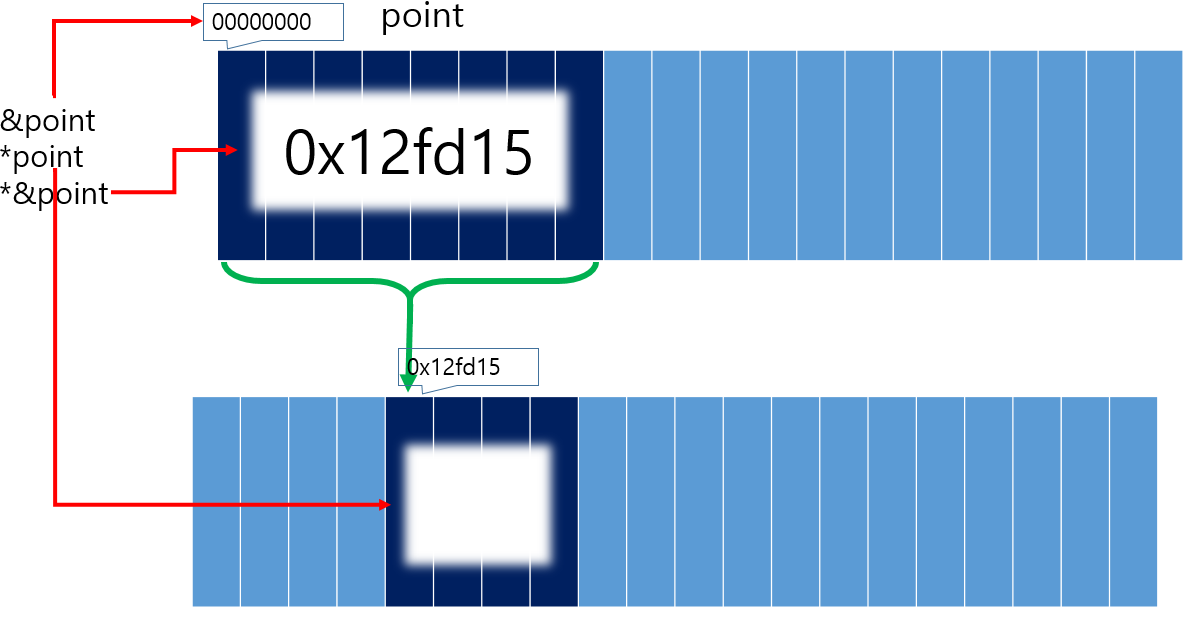


이 정도 배웠으면, 주소에 대한 개념이 어느 정도 잡혔을 것이다. 그렇다면 본격적으로 포인터에 대해 알아보자.

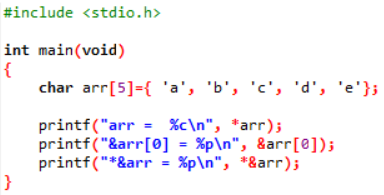
7-1-2 포인터의 정의

포인터를 사전적으로 해석하면, **‘가리키는 것’**이다. 앞서 나온 비유처럼 배달부 아저씨가 편지를 배달할 때, 집 주소를 보고 찾아간다. 주소를 보고 따라가면 그 집이 누구 집인지, 집 안에 무엇이 있는지 알 수 있다. 이처럼 포인터는 ‘주소가 가리키는 곳에 있는 값’을 알려준다. 따라서 포인터 변수 안에는 변수의 위치를 알려주는 주소가 들어가게 된다. 🡪 포인터는 **배달부 아저씨**야. 배달부 아저씨한테 주소를 알려주면 아저씨는 주소를 정확히 찾아 배달을 한다.





포인터를 사용할 때는 연산자 \*(애스터리스크, asterisk)를 사용한다. 따라서 \*는 주소를 가르쳐주는 연산자 &와 반대의 관계를 가진다. 그래서 위와 같이 point라는 변수가 있다고 가정하면, &point는 point라는 변수의 주소를, \*point는 point 변수가 가리키는 메모리 안에 있는 값을 나타내고, 둘을 동시에 쓴 \*&point는 연산자끼리 서로 상쇄되어 point랑 똑같은 역할을 하게 된다. 코드를 통해 하나씩 확인해보자.



7-2 포인터 변수의 선언과 사용

7-2-1 포인터 변수의 선언

포인터 변수의 선언은 다음과 같이 한다.

(데이터 타입)\* (포인터 변수 이름) =NULL;

Ex) int \*point = NULL;

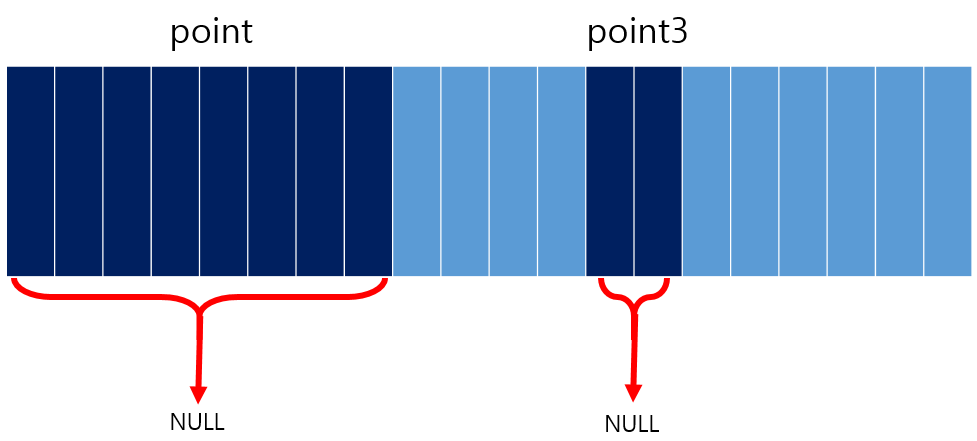
double \*point2 = NULL;

char \*point3 = NULL;

* 이 때 \*는 (데이터 타입) \*(포인터 변수 이름), (데이터 타입) \*(포인터 변수 이름) 두 가지 방법 중 어떤 방법으로 선언되어도 상관없다.
* 그러나 여기서 주의할 점은, 두 변수를 한번에 선언할 때 int \*point, point2;라고 선언을 하면 point는 포인터변수, point2는 일반변수로 인식이 된다. 마찬가지로 int point, \*point2;라고 선언하면 point는 일반변수, point2는 포인터변수가 된다.

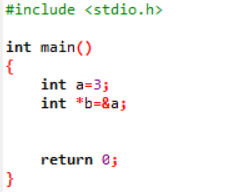
int \*point는 int형 변수의 주소를 저장하는 포인터 변수이고, double \*point2는 double형 변수의 주소를, char \*point3는 char형 변수의 주소를 저장하는 포인터 변수이다.

엄밀히 말하면 포인터 변수도 변수이기 때문에, 초기화해주지 않으면 쓰레기 값이 들어간다. 따라서 아무 주소도 저장하지 않겠다는 표시인, NULL문자를 넣어주어서 초기화 해주는 것이 좋다

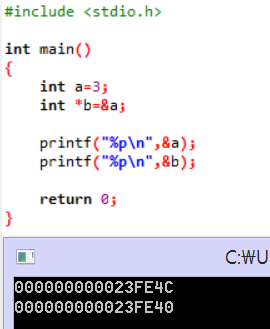


7-2-2 포인터 변수의 사용

자 이제, 포인터 변수를 선언했으니 변수 안에 값을 한 번 넣어보자. 포인터에는 주소 값이 들어간다! 또, 우린 변수의 주소 값을 나타내주는 연산자가 &라는 것을 알고 있다. 따라서 포인터 변수 안에 값을 넣을 때는 아래 코드와 같이 써주면 된다..

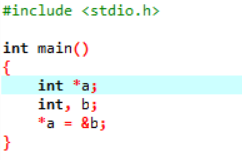


차례대로 해석해보자. 처음에 a라는 이름의 변수를 선언(상자 생성)한다. 그 다음에 int형 주소를 넣을 수 있는 포인터 변수 b를 선언(주소 넣는 상자 생성)한다. 그리고 동시에 a의 주소를 넣어 준다. 주소가 어떻게 생겨먹었는지, 또 값이 제대로 들어갔는지 확인해보기 위해 몇 줄을 더 추가한 아래 코드도 한 번 보자.



결과적으로 a의 주소는 위 그림과 같이 나오고, \*b는 b에 들어있는 a의 주소를 따라 a 안에 들어있는 3을 출력하게 된다.

참고로 포인터 변수에서 \* 연산자를 사용할 때 띄어쓰기를 해줘도 상관이 없기 때문에 아래와 같은 실수를 하게 되는 경우가 있다.



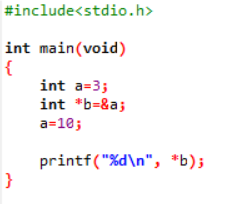
‘\*a= &b;’ 라는 표기는 잘못된 사용이다. \*a는 이미 값을 의미하고 있는데(l-value), 값에 다른 값을 대입하는 것은 불가능하다. 대입은 l-value(좌항)이 변수일 때만 가능하다.

변수의 이름이 \*a이 아니라 a이므로 ‘a = &b;’ 라고 쓰는 것이 맞는 표현이다.

7-2-3 포인터 변수 사용 시 유의 사항

포인터 변수에 값을 넣을 때 주의해야 할 점이 있는데, 주소를 내 마음대로 설정하거나 바꿔선 안 된다는 것이다. 주소 값은 컴퓨터가 특정한 곳을 정해서 넣어주는 것이기 때문에, 마음대로 설정해버리면 에러가 나게 된다. 따라서 포인터 변수에 값을 넣을 때는 임의의 주소값 할당이 아닌 ‘&변수명’ 형태로 이루어져야 한다.

(그림: 아잉 여기는…. 안돼 하고 있는… 컴터가 몸 가리고 있는 모습)

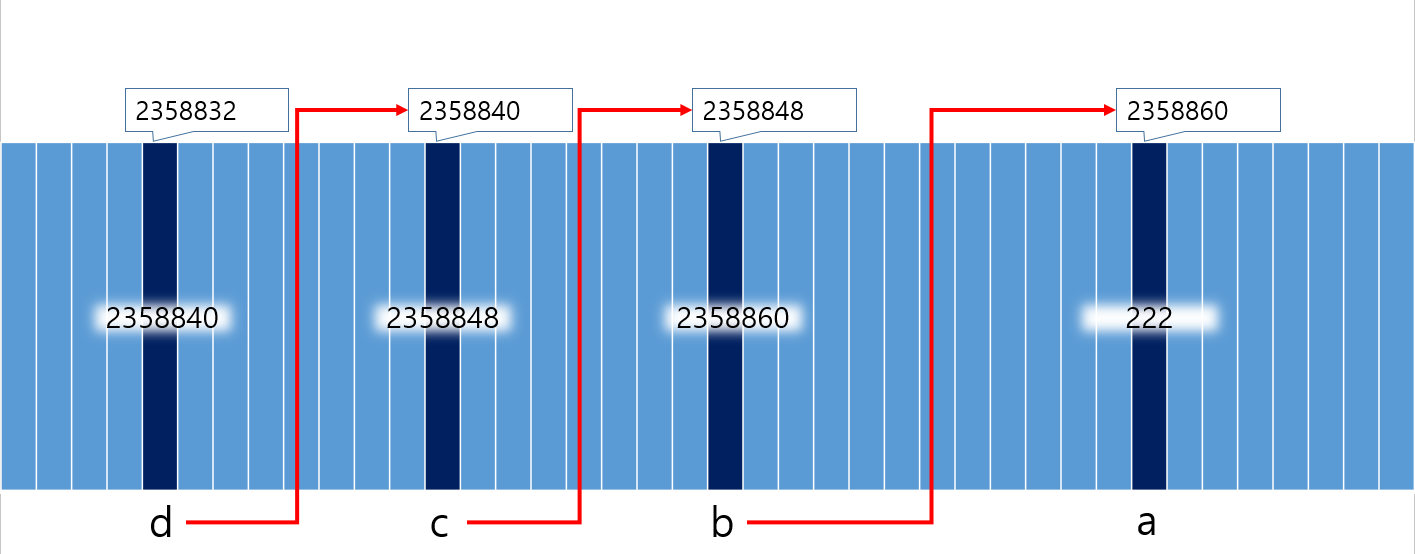


C:\Users\배지노\Desktop\15.JPG

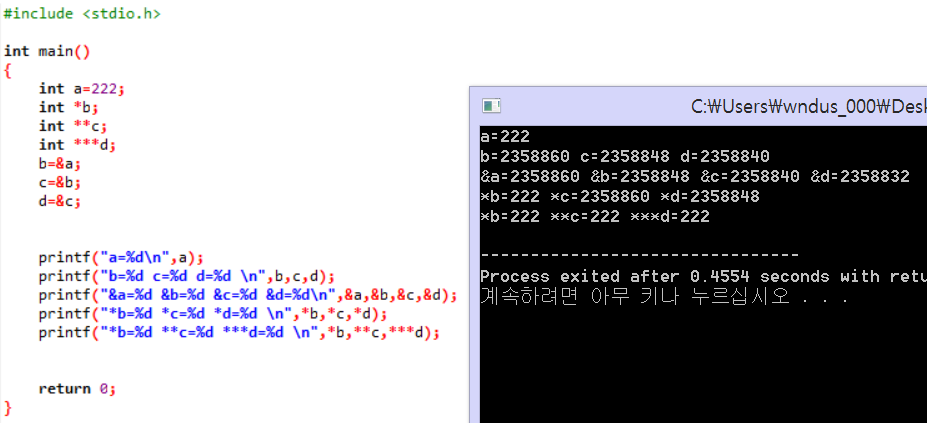
7-3 다중 포인터

추리 소설을 보면, 한 가지 힌트를 통해 사건을 해결하는 이야기도 있지만, 사건이 꼬리에 꼬리를 물어서 차례차례 정보를 찾아내는 사건도 있다. 한 힌트의 답을 찾으면 그 답을 통해 다음 정보에 대한 힌트를 얻을 수 있는 것이다. 이러한 구조를 포인터에서도 볼 수 있다. 바로 다차원 포인터이다.

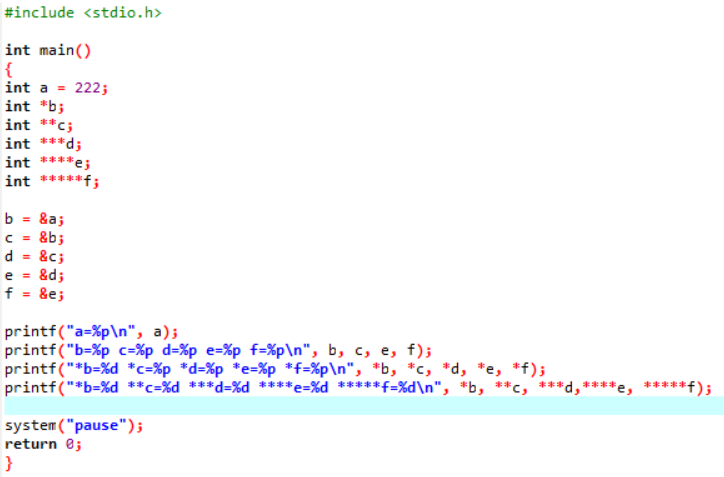
\* 연산자가 여러 개 붙어있는 \*\* 나 \*\*\*를 다중 포인터라고 한다. (\*\*\*는 안 쓰임) 한 포인터 변수 안에 주소 값이 들어 있어서 그 주소 값을 찾아가 봤더니, 또 다른 주소가 있는 것이다. 이렇게 다차원 포인터는 꼬리에 꼬리를 물게 된다. 그림으로 확인 하면 더 쉽게 이해할 수 있을 것이다. 아래의 그림은 3중포인터를 표현한 것이다.

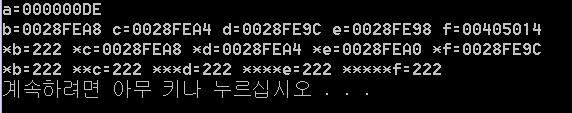


위의 그림을 코드로 변환시키면 아래와 같이 표현할 수 있다.



더 나아가, 5중 포인터를 사용할 수도 있다.





조금만 더 자세히 보면, 일차원 포인터 변수는 일반적인 변수의 주소를 저장하고, 이차원 포인터 변수는 일차원 포인터 변수의 주소를 저장한다는 것을 알 수 있다. 그리고 3차원 포인터 변수는 2차원 포인터 변수의 주소를 저장하는 것이다. 이 개념은 다음에 배울 포인터와 이차원 배열의 관계에서 중요하므로 잘 숙지하도록 하자.

7-4 동적 할당

7-4-1 동적 할당의 필요성: 메모리를 자유롭게 활용하자

지금까지 포인터의 기초 개념에 대해 알아보았다. 포인터의 심화 개념에 대해서는 다음 장에서 더 알아보기로 하고, 이번 장의 마지막에서는 ‘동적 할당’에 대해 알아보도록 한다. 독자들이 이 책을 처음 피고 이번 7장까지 배운 개념들을 떠올려 보자. 일반적인 변수, 배열, 구조체 등 수 많은 변수에 대해 공부하던 것을 기억할 것이다. 동적 할당을 하는 이유는 이러한 변수들을 더욱 자유자재로 활용할 수 있기 위함이다. 동적할당의 대표적인 함수는 malloc함수인데 malloc을 사용하기 위해서는 <stdlib.h>라이브러리가 필요하다.

7-4-2 동적 할당과 해제

**<할당하는 기본 문법> 포인터 변수 = malloc(sizeof(앞 변수의 자료형) \* 원하는 크기);**



원하는 크기에는 일반 변수일 경우 1, 배열로 만들고 싶을 경우 원하는 크기 수 만큼 곱한다.

**<해제하는 기본 문법> free(위에서 동적 할당 했던 포인터 변수)**



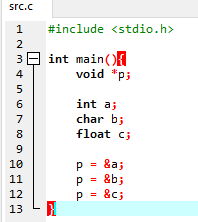
동적할당 사용시 프로그램이 종료되기 전에 반드시 free함수를 통해서 해제를 해주어야한다는 점 잊지 말자.

7-4-3 동적 할당에서의 유의 사항

할당한 범위를 벗어나는 참조를 금지해야 한다. 다음 장에서 배울 배열을 포인터처럼 이용하는 경우 특히 주의해야 한다.

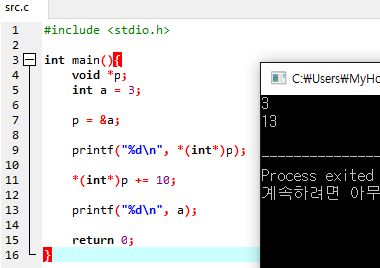
@ 더 알아보기: void 포인터

포인터의 기초적인 내용들을 두루 다루었던 이번 장의 마지막으로 독자 여러분들이 알고 갔으면 하는 것이 있다. 모든 변수에 자료형을 지정할 때 int, char, short, float, double 등 수 많은 자료형들이 있지만 자료형을 지정하지 않는 경우는 없다. 그렇지만 포인터 변수는 조금 다르다. 포인터 변수의 경우 자료형을 지정하지 않고 선언하는 방법이 있는데, 이를 void 포인터라고 부른다. 예제를 통해 살펴보도록 하자.



포인터를 void형으로 선언하면 어떤 자료형의 변수이든 간에 그 변수의 주소를 void 포인터에 저장할 수가 있다. 이를 통해 알 수 있는 점은 뒷장의 함수 단원에서 배우겠지만, 주소 값은 알지만 자료형은 모르는 경우가(컴파일러의 입장에서) 종종 있는데 이러한 경우에 사용하는 것이 void 포인터이다.

다만 void 포인터는 바로 사용하지는 못한다. 간접 참조 연산자(\*)를 이용할 수가 없다는 말이다. 그러면 도대체 이 포인터는 무용지물이 아닌가! 왜 쓰는 것인가? 라고 생각하는 독자가 있을 수도 있는데, 우선 침착하시고 어떻게 사용하는지도 알려줄 테니 차분히 더 보기를 바란다.



위의 소스 코드를 보면 p에다가 앞에 (int\*)를 해줌으로써 void\* 형을 int\* 형으로 명시적 형 변환을 해 주었다. 그리고 여기에다 앞에 간접 참조 연산자(\*)를 붙임으로써 드디어 a 값에 접근할 수 있게 되었다!