



혼.공.C CHAPTER.9

포인터란?

“포인터 = 주소”

포인터란?

포인터는 다른 말로 포인터 변수라고 한다.
변수는 어떤 값을 저장하는 것이니 포인터는 주소 값을 저장하는 변수이다.
이때 주소란 변수가 할당된 메모리 공간의 시작주소이다.

```
#include <stdio.h>
```

```
int main(void){  
    int a;  
    double b;  
    char c;  
  
    printf("int형 변수의 주소 : %u\n", &a);  
    printf("double형 변수의 주소 : %u\n", &b);  
    printf("char형 변수의 주소 : %u\n", &c);  
  
    return 0;  
}
```

Microsoft Visual Studio 디버그 콘솔

```
int형 변수의 주소 : 13106460  
double형 변수의 주소 : 13106444  
char형 변수의 주소 : 13106435
```

주소
연산자
& 변수명

포인터란?

```
#include <stdio.h>
```

```
int main(void){
```

```
    int a;
```

```
    double b;
```

```
    char c;
```

```
    printf("int형 변수의 주소 : %u\n", &a);
```

```
    printf("double형 변수의 주소 : %u\n", &b);
```

```
    printf("char형 변수의 주소 : %u\n", &c);
```

```
    return 0;
```

```
}
```

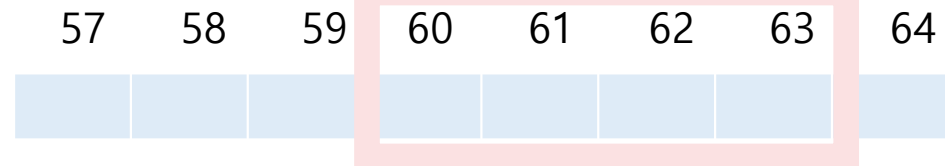
Microsoft Visual Studio 디버그 콘솔

int형 변수의 주소 : 13106460

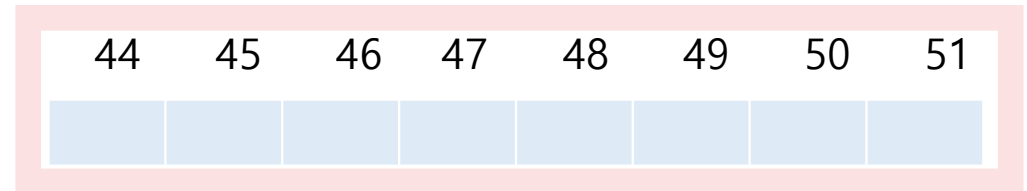
double형 변수의 주소 : 13106444

char형 변수의 주소 : 13106435

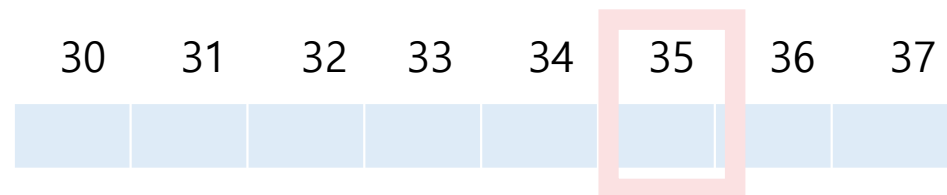
Int형 4바이트



double형 8바이트



char형 1바이트



포인터란?

주소를 저장할 포인터도 변수처럼 선언해서 사용하면 된다.
일반 변수와의 차이점은 포인터는 변수 앞에 *(간접 참조 연산자)를 붙여준다.
이때 자료형은 저장할 주소가 어떤 변수의 주소인지 변수의 자료형을 저장한다.

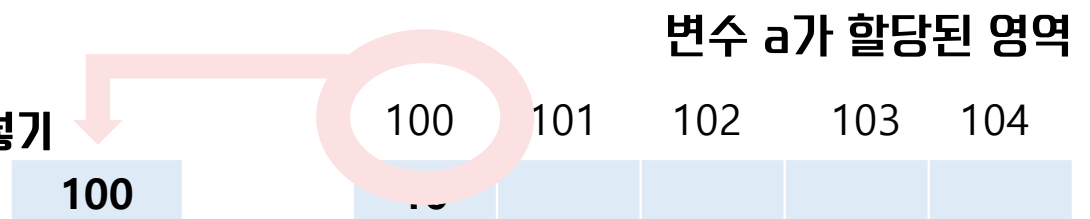
```
#include <stdio.h>
```

```
int main(void){
    int a;
    int *pa; //포인터 변수

    pa = &a; // 포인터 pa에 a 변수의 시작 주소 넣기
    *pa = 10;

    printf("포인터로 a값 출력 : %d\n", *pa);
    printf("변수명으로 a값 출력 : %d\n", a);

    return 0;
}
```



Pa -> a

Microsoft Visual Studio 디버그 콘솔

```
포인터로 a값 출력 : 10
변수명으로 a값 출력 : 10
```

포인터란?

```
#include <stdio.h>
```

```
int main(void){  
    int a=10, b=15, total;  
    double avg;  
    int* pa, * pb;  
    int* pt = &total;  
    double* pg = &avg;
```

```
    pa = &a;  
    pb = &b;
```

```
    *pt = *pa + *pb;  
    *pg = *pt / 2.0;
```

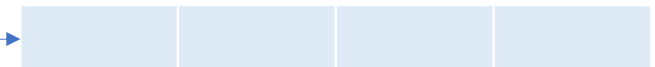
```
    printf("두 정수의 값 : &d, %d\n", *pa, *pb);  
    printf("두 정수의 합 : %d\n", *pt);  
    printf("두 정수의 평균 : %.1f\n", *pg);
```

Microsoft Visual Studio 디버그 콘솔

```
두 정수의 값 : &d, 10  
두 정수의 합 : 25  
두 정수의 평균 : 12.5
```

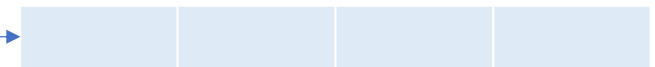
100 101 102 103 a

pa 100



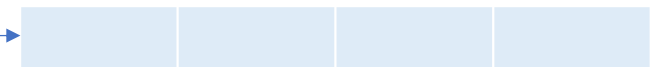
200 201 202 203 b

pb 200



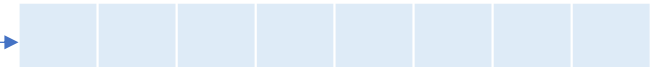
300 301 302 303 total

pt 300



400 401 402 403 404 405 406 407 avg

pg 400



포인터란?

```
#include <stdio.h>

int main(void){
    int a = 10, b = 20;
    const int* pa = &a;

    printf("변수 a의 값 : %d\n", *pa);
    pa = &b;
    printf("변수 b의 값 : %d\n", *pa);
    pa = &a;
    a = 20;
    printf("변수 a 값 : %d\n", *pa);

    return 0;
}
```

Microsoft Visual Studio 디버깅 콘솔 출력:

```
변수 a의 값 : 10
변수 b의 값 : 20
변수 a 값 : 20
```

Const는 변수 앞에 붙이면 값을 변경하지 못하도록 하며 해당 변수를 상수 취급하게 된다.
변수의 초기값이 변하지 않게 할 때 쓴다.

위 예시의 경우
*pa는 a의 주소의 값을 받아왔는데 앞에 const가 붙어 변수의 값을 변경하지는 못한다.
즉, *pa의 변수 값은 상수가 되었기 때문에 변경할 수 없다.

포인터란?

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int num = 10;
6      int *ptr1 = &num;
7      const int *ptr2 = &num;
8
9      *ptr1 = 20;
10     num = 30;
11
12     *ptr2 = 40;
13
14     return 0;
15 }
```

Const로 선언하지 않은 ptr1은 *ptr1을 이용해 20을 대입할 수 있다.

하지만 const로 선언한 ptr2는 상수화되어 변수의 값을 변경할 수 없다. 따라서 40을 대입하려고 하자 오류가 난다.

Main.c: In function 'main':

Main.c:12:15: error: assignment of read-only location '*ptr2'

```
12 |         *ptr2 = 40;
```

```
|         ^
```

make: *** [cmd] Error 1

포인터 정리

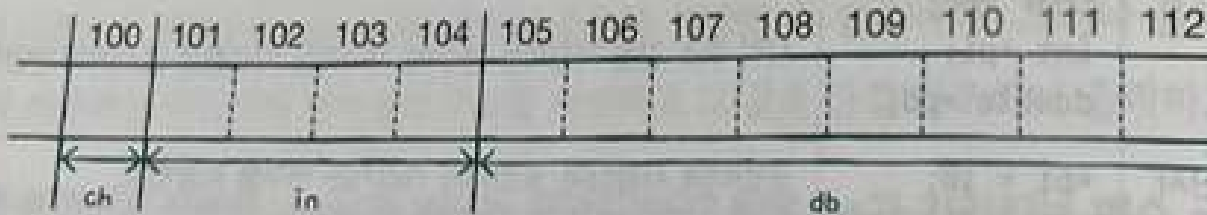
- 포인터는 메모리를 사용하는 또 다른 방법이다.
- 주소 연산자 &로 변수가 할당된 메모리의 위치를 확인한다.
- 포인터로 가리키는 변수를 사용할 때는 간접 참조 연산자 *를 쓴다.
- Const는 pa가 가리키는 변수 a는 pa를 간접 참조하여 바꿀 수 없다는 뜻이다.

포인터 정리

구분	사용 예	기능
주소 연산자	<code>int a;</code> <code>&a;</code>	변수 앞에 붙여 사용하며, 변수가 할당된 메모리의 시작 주소 값을 구한다.
포인터	<code>char *pc;</code> <code>int *pi;</code> <code>double *pd;</code>	시작 주소 값을 저장하는 변수며, 가리키는 자료형을 표시하여 선언한다.
간접 참조 연산자	<code>*pi=10;</code>	포인터에 사용하며, 포인터가 가리키는 변수를 사용한다.

확인 문제

```
char ch = 'A';
int in = 10;
double db = 3.4;
```



수식	&ch	&in	&db	*&ch	*&in	*&db
값						

```
#include <stdio.h>
```

```
void swap(int x, int y);
```

```
int main(void)
```

```
{
```

```
char ch = 'A';
```

```
int in = 10;
```

```
double db = 3.4;
```

```
printf("&ch 값 : %d\n", *&ch);
```

```
printf("&in의 값 : %d\n", *&in);
```

```
printf("&db 값 : %d\n", *&db);
```

```
return 0;
```

```
}
```

Microsoft Visual Studio 디버그 콘솔

```
*&ch 값 : 65
```

```
*&in의 값 : 10
```

```
*&db 값 : 858993459
```

확인 문제

3. 다음 코드의 실행결과를 작성하세요.

```
int a = 10;  
int *p = &a;  
*p = 20;  
printf("%d", a);
```

```
#include <stdio.h>
```

```
void swap(int x, int y);
```

```
int main(void)
```

```
{
```

```
    int a = 10;
```

```
    int* p = &a;
```

```
    *p = 20;
```

```
    printf("%d", a);
```

```
    return 0;
```

```
}
```

Microsoft Visual Studio 디버그

20

포인터 이해하기

주소 -> 변수에 할당된 메모리 공간의 시작 주소 값 자체

포인터 -> 그 값을 저장하는 또 다른 메모리 공간

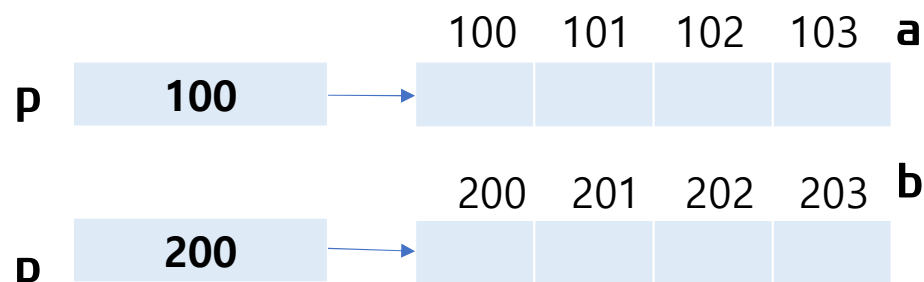
따라서 특정 변수의 주소 값은 바뀌지 않지만 포인터는 다른 주소를 대입해 바꿀 수 있다.

```
Int a,b;    // 일반 변수 선언
```

```
Int *p      // 포인터 선언
```

```
p = &a;     // p가 a를 가리키도록 설정
```

```
P = &b;     // p가 변수 b를 가리키도록 바꿈
```



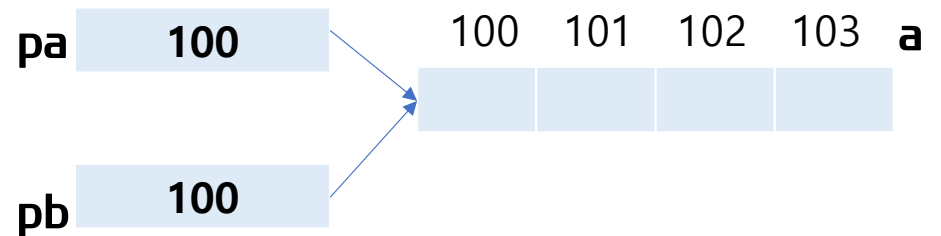
포인터 이해하기

주소 -> 변수에 할당된 메모리 공간의 시작 주소 값 자체

포인터 -> 그 값을 저장하는 또 다른 메모리 공간

따라서 특정 변수의 주소 값은 바뀌지 않지만 포인터는 다른 주소를 대입해 바꿀 수 있다.

```
Int a; // 일반 변수 선언
Int *pa, *pb; // 가리키는 자료형이 같음
Pa = pb=&a; // pa와 pb에 a의 주소 저장
```



포인터 이해하기

```
char ch;  
int in;  
double db;  
  
char* pc = &ch;  
int* pi = &in;  
double* pd = &db;
```

포인터도 저장 공간이라
크기가 있다.
모든 주소와 포인터는
자료형과 상관 없이
크기가 같다.

```
&ch;  
&in;  
= &db;
```

```
r형 변수 주소 크기 : %d\n", sizeof(&ch));  
형 변수 주소 크기 : %d\n", sizeof(&in));  
ble형 변수 주소 크기 : %d\n", sizeof(&db));  
r형 포인터 크기 : %d\n", sizeof(pc));  
형 포인터 크기 : %d\n", sizeof(pi));  
ble형 포인터 크기 : %d\n", sizeof(pd));  
r형 포인터가 가리키는 변수의 크기 : %d\n", sizeof(*pc));  
형 포인터가 가리키는 변수의 크기 : %d\n", sizeof(*pi));  
ble형 포인터가 가리키는 변수의 크기 : %d\n", sizeof(*pd));
```

문제 검색되지 않음

선택(S): 디버그

```
.exe'(Win32): 'C:\Users\whu612\Desktop\c언어\9단원\Pro  
.exe'(Win32): 'C:\Windows\SysWow64\ntdll.dll'을(를)  
.exe'(Win32): 'C:\Windows\SysWow64\kernel32.dll'을(를)  
.exe'(Win32): 'C:\Windows\SysWow64\KernelBase.dll'을(를)  
.exe'(Win32): 'C:\Windows\SysWow64\vcruntime140d.dll'
```

Microsoft Visual Studio 디버그 콘솔

```
char형 변수 주소 크기 : 4  
int형 변수 주소 크기 : 4  
double형 변수 주소 크기 : 4  
char형 포인터 크기 : 4  
int형 포인터 크기 : 4  
double형 포인터 크기 : 4  
char형 포인터가 가리키는 변수의 크기 : 1  
int형 포인터가 가리키는 변수의 크기 : 4  
double형 포인터가 가리키는 변수의 크기 : 8
```

포인터 이해하기

규칙

- 1) 포인터는 가리키는 변수의 형태가 같을 때 대입할 수 있다.
- 2) 형 변환을 사용한 포인터의 대입은 언제나 가능하다.

다른 변수의 형태일때
대입을 하면 다음과 같이
할당되지 않은 영역까지 나온다.

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int a = 10;
```

```
    int* p = &a;
```

```
    double* pd;
```

```
    pd = p;
```

```
    printf("%lf\n", *pd);
```

```
    return 0;
```

```
}
```

Microsoft Visual Studio 디버그 콘솔

-92559592117432107884277659021957555520241347761778250032873472.000000

포인터 이해하기

규칙

- 1) 포인터는 가리키는 변수의 형태가 같을 때 대입할 수 있다.
- 2) 형 변환을 사용한 포인터의 대입은 언제나 가능하다.

```
int main(void)
{
    double a = 3.4;
    double* pd = &a;
    int* pi;
    pi = (int*)pd;

    printf("%d\n", pi);

    return 0;
```

Microsoft Visual Studio 디버거

17824112

포인터 이해하기

* 포인터 사용 이유?!

변수를 사용하는 가장 쉬운 방법은 이름을 쓰는 것이다.

포인터를 쓰는 이유는 주로 임베디드 프로그래밍을 할 때 메모리에 직접 접근하거나 동적 할당된 메모리를 사용하는 경우이다.

* 임베디드 프로그래밍 ?!

특정 기능을 제어하기 위한 프로그램으로 하드웨어를 제어하는 소프트웨어를 만들어내는 일을 하는 것을 임베디드 프로그래밍이라고 한다. 예로 정수, 냉수를 구분해 물이 나오게 하는 것이다.

포인터 이해하기

```
void swap(int* pa, int* pb);
```

```
int main(void)
{
    int a = 10, b = 20;
```

포인터 없이 두 변수의 값을 바꾸는 것은 불가능하 !!

```
void swap(int* pa, int* pb)
{
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
```

```
a:20, b:10
C:\Users\whu612\Desktop
디버깅이 중지될 때
하도록 설정합니다
이 창을 닫으려면
```

포인터 이해하기

```
void swap(void);

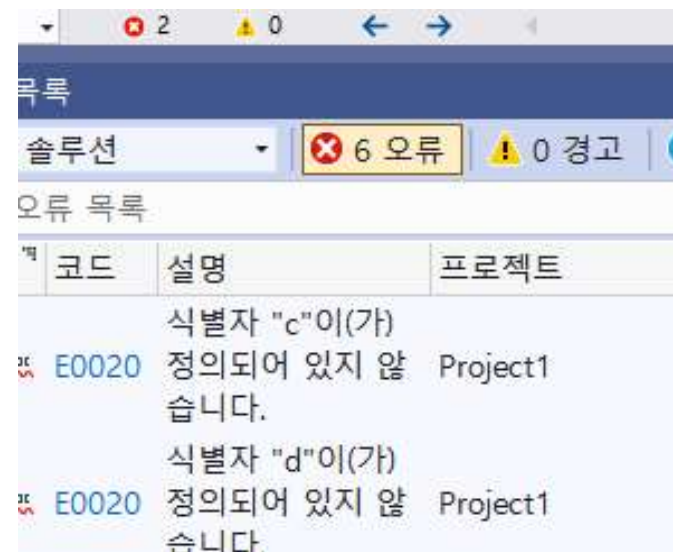
int main(void)
{
    int c=10, d=20;

    swap();
    printf("c:%d, d:%d\n", c, d);

    return 0;
}

void swap(void)
{
    int temp;

    temp = c;
    c = d;
    d = temp;
}
```



포인터 이해하기

```
void swap(int x, int y);

int main(void)
{
    int a=10, b=20;

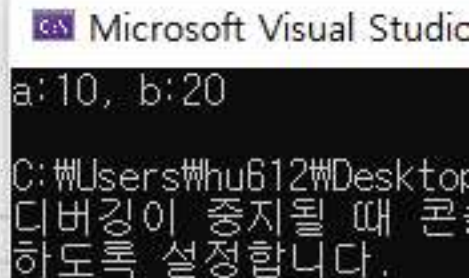
    swap(a,b);
    printf("a:%d, b:%d\n", a, b);

    return 0;
}

void swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```

Swap 함수 안에서 복사본으로 값을 주는 것이기 때문에 main 함수로 갈 수 없다.



Microsoft Visual Studio
a:10, b:20
C:\Users\hu612\Desktop
디버깅이 중지될 때 콘솔
화면이 사라집니다.

포인터 정리

- 주소와 포인터는 상수와 변수의 차이가 있다.
- 포인터의 크기는 주소의 크기와 같다.
- 포인터에 주소를 저장할 때는 가리키는 자료형이 같아야 한다.
- 포인터의 주요 기능 중 하나는 함수 간에 효과적으로 데이터를 공유하는 것이다.

포인터 정리

구분	변수 a 사용	포인터 pa 사용
대입 연산자 왼쪽	<code>a = 10;</code>	<code>*pa = 10;</code>
대입 연산자 오른쪽	<code>b = a;</code>	<code>b = *pa;</code>
피연산자	<code>a + 20;</code>	<code>*pa + 20;</code>
출력	<code>printf("%d", a);</code>	<code>printf("%d", *pa);</code>
입력	<code>scanf("%d", &a);</code>	<code>scanf("%d", &*pa);</code> <code>scanf("%d", pa);</code>

포인터 정리

구분	사용 예	기능
포인터	<pre>int a,b; int *p = &a; p = &b;</pre>	포인터는 변수이므로 그 값을 다른 주소로 바꿀 수 있다.
포인터 크기	<pre>int *p; sizeof(p)</pre>	포인터의 크기는 컴파일러에 따라 다를 수 있으며 sizeof 연산자로 확인 한다.
포인터 대입 규칙	<pre>int *p; double *pd; pd = p;(x)</pre>	포인터는 가리키는 자료형이 일치할 때만 대입한다.

확인 문제

3. 다음 코드의 실행결과를 작성해보세요.

```
int a = 10, b = 20;
int *pa = &a, *pb = &b, *pt;
pt = pa;
pa = pb;
pb = pt;
printf("%d, %d", *pa, *pb);
```

```
#include <stdio.h>

int main(void)
{
    int a = 10, b = 20;
    int* pa = &a, *pb = &b, *pt;

    pt = pa;
    pa = pb;
    pb = pt;
    printf("%d, %d", *pa, *pb);
    return 0;
}
```

Microsoft Visual Studio 디버거 콘솔

20, 10

확인 문제

2. 주소 값의 크기가 4바이트일 때, sizeof 연산의

```
char *pc;  
double *pd;
```

- ① sizeof(pc) ② sizeof(pd)
- ③ sizeof(*pc) ④ sizeof(*pd)

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
char* pc;
```

```
double* pd;
```

```
printf(" pc 포인터 크기 : %d\n", sizeof(pc));
```

```
printf(" pd 포인터 크기 : %d\n", sizeof(pd));
```

```
printf(" pc 포인터가 가리키는 변수 크기 : %d\n", sizeof(*pc));
```

```
printf(" pd 포인터가 가리키는 변수 크기 : %d\n", sizeof(*pd));
```

```
return 0;
```

```
}
```

Microsoft Visual Studio 디버그 콘솔

```
pc 포인터 크기 : 4  
pd 포인터 크기 : 4  
pc 포인터가 가리키는 변수 크기 : 1  
pd 포인터가 가리키는 변수 크기 : 8
```

끝