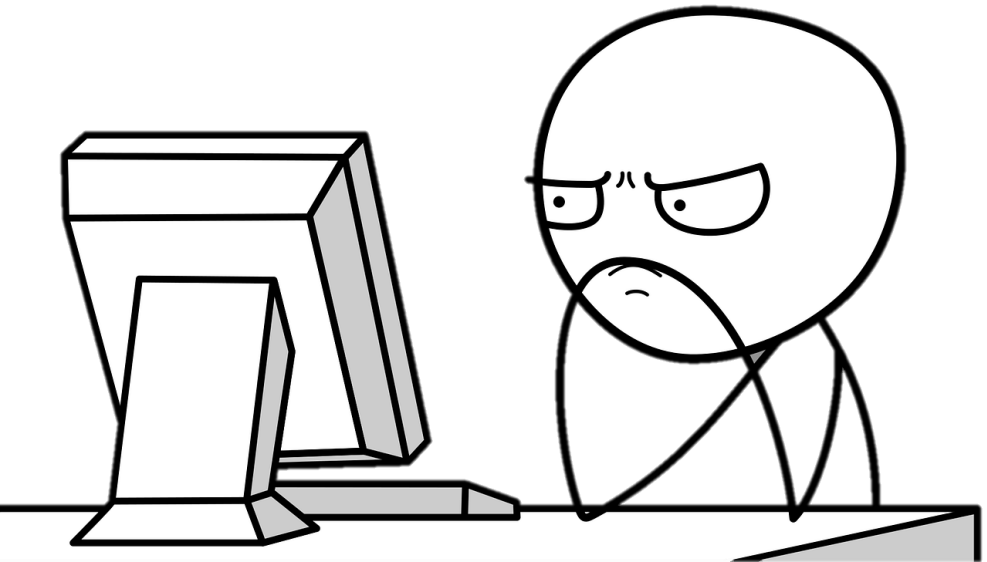

컴퓨터 프로그래밍
Computer Programming

08
포인터 기초

목차

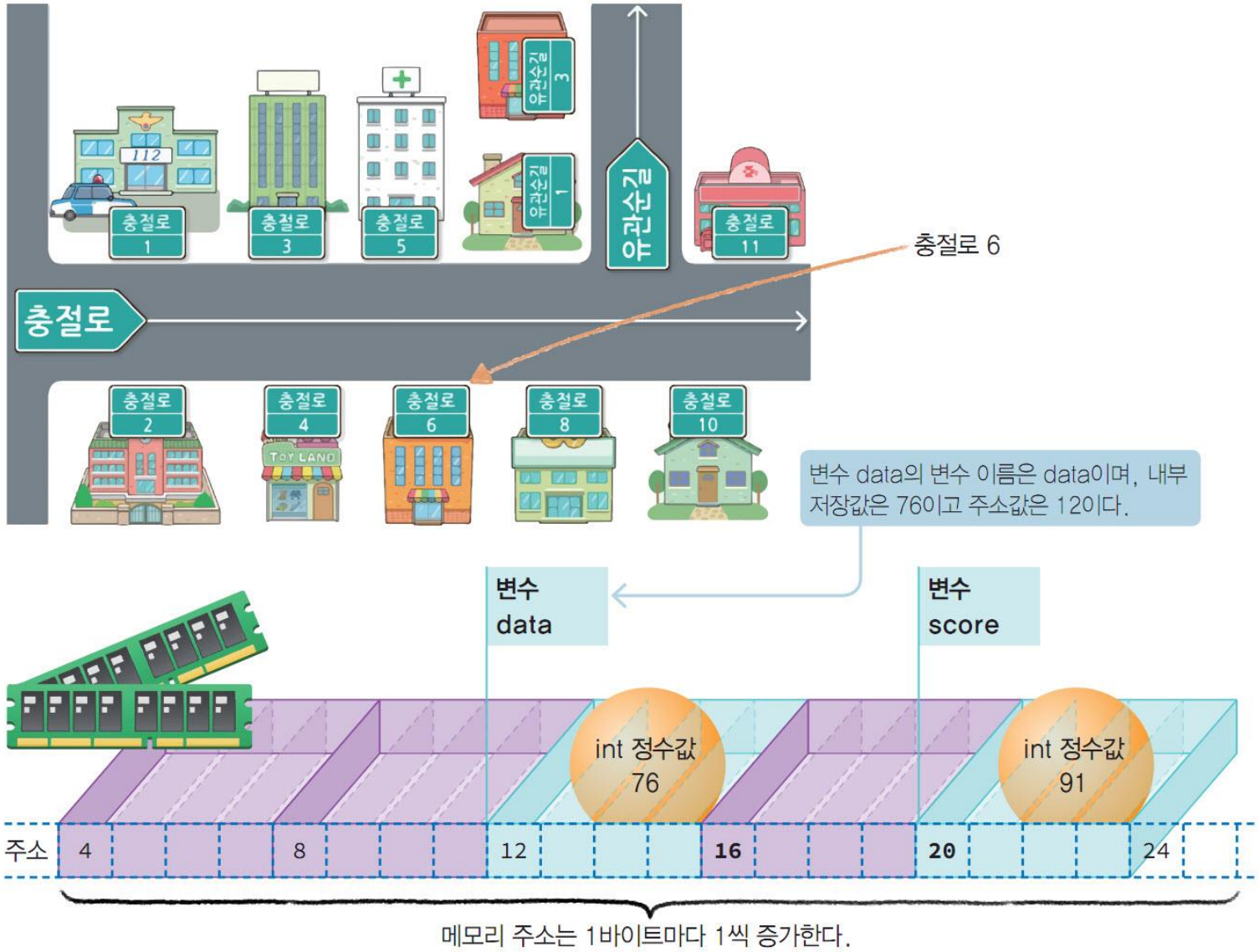
1. 포인터 변수와 선언



주소 개념

- 주소address
  - 메모리 공간은 8비트인 1 바이트마다 순차적인 고유한 번호
  - 메모리 주소는 저장 장소인 변수이름과 함께 기억 장소를 참조하는 또 다른 방법
    - '렉슬아파트'와 같이 아파트이름이 변수이름
    - '선릉로 888'과 같이 도로명과 번호가 메모리 주소
  - 메모리 주소가 왜 필요하지요?
    - 보다 편리하고 융통성 있는 프로그램이 가능
      - 주소 정보를 이용하여 주소가 가리키는 변수의 값을 참조 가능
      - 주소 정보의 이전 또는 이후의 이웃한 저장 공간의 값도 쉽게 참조 가능

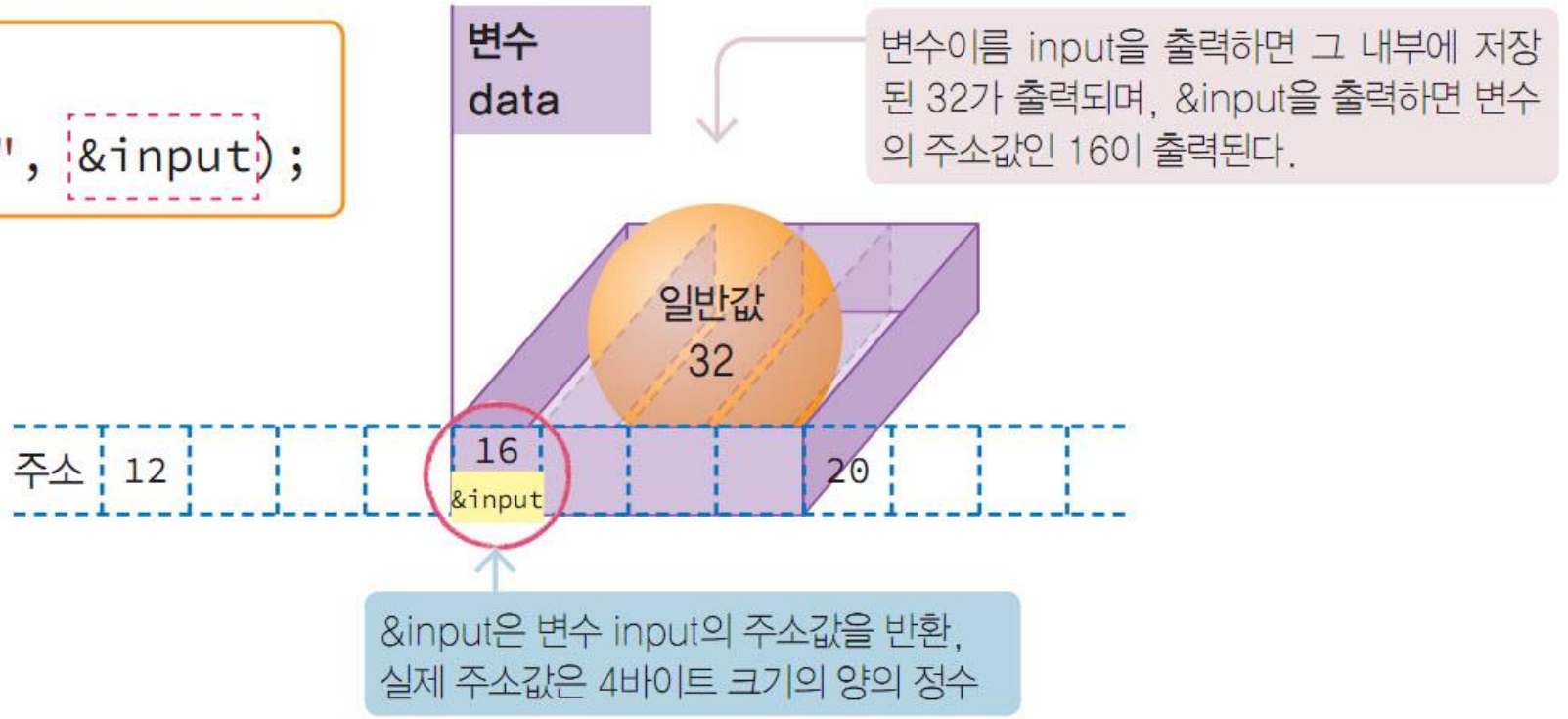
주소 개념


충절로
유관순길
충절로 1
충절로 3
충절로 5
유관순길 3
유관순길 1
충절로 11
충절로 6
충절로 2
충절로 4
충절로 6
충절로 8
충절로 10
변수 data의 변수 이름은 data이며, 내부 저장값은 76이고 주소값은 120이다.
변수 data
변수 score
int 정수값 76
int 정수값 91
주소
4
8
12
16
20
24
메모리 주소는 1바이트마다 1씩 증가한다.

주소연산자 &

- 함수 scanf()를 사용하면서 인자를 '&변수이름'으로 사용
  - 바로 &(ampersand)가 피연산자인 변수의 메모리 주소를 반환하는 주소연산자
    - 함수 scanf()에서 입력값을 저장하는 변수의 주소값이 인자의 자료형
    - 함수 scanf()에서 일반 변수 앞에는 주소연산자 &를 사용

```
int input;
    scanf("%d", &input);
```



Source Code #01: address.c

```c
// file: address.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int input;

    printf("정수 입력: ");
    scanf("%d", &input);
    printf("입력 값: %d\n", input);
    printf("주소 값: %u(10진수), %p(16진수)\n", (int)&input, &input);
    printf("주소 값: %d(10진수), %#X(16진수)\n", (unsigned)&input, (int)&input);
    printf("주소 값 크기: %d\n", sizeof(&input));

    return 0;
}
```

```
정수 입력: 100
입력 값: 100
주소 값: 5241288(10진수), 004FF9C8(16진수)
주소 값: 5241288(10진수), 0X4FF9C8(16진수)
주소 값 크기: 4
```

- 변수의 값과 주소값을 출력
- 변수의 주소값 출력
  - 형식제어문자 %u 또는 %d로 직접 출력
  - 최근 비주얼 스튜디오에서는 경고가 발생하니 주소값을 int 또는 unsigned로 변환하여 출력
- 만일 16진수로 출력
  - 형식제어문자 %p를 사용
- & 연산자는 '&변수'와 같이 피연산자 앞에 위치하는 전위연산자로 변수에만 사용 가능
  - '&32'와 '&(3+4)'은 오류

포인터 변수

- 주소값을 저장하는 변수
  - 변수의 주소값은 반드시 포인터 변수에 저장
  - 일반 변수에는 일반 자료 값이 저장
- 일반 변수와 구별되며 선언방법이 다름

메모리 주소를 저장하는 포인터 변수

포인터 변수 선언

- 선언 방법
  - 포인터 변수 선언에서 자료형과 포인터 변수 이름 사이에 연산자 *(asterisk)를 삽입
  - ptrint, ptrshort, ptrchar, ptrdouble은 모두 포인터 변수
    - 간단히 포인터라고도 부름
  - 예로 'int *ptrint' 선언
    - 'int 포인터 ptrint'라고 읽도록
  - 변수 자료형이 다르면
    - 그 변수의 주소를 저장하는 포인터의 자료형도 반드시 달라야 함

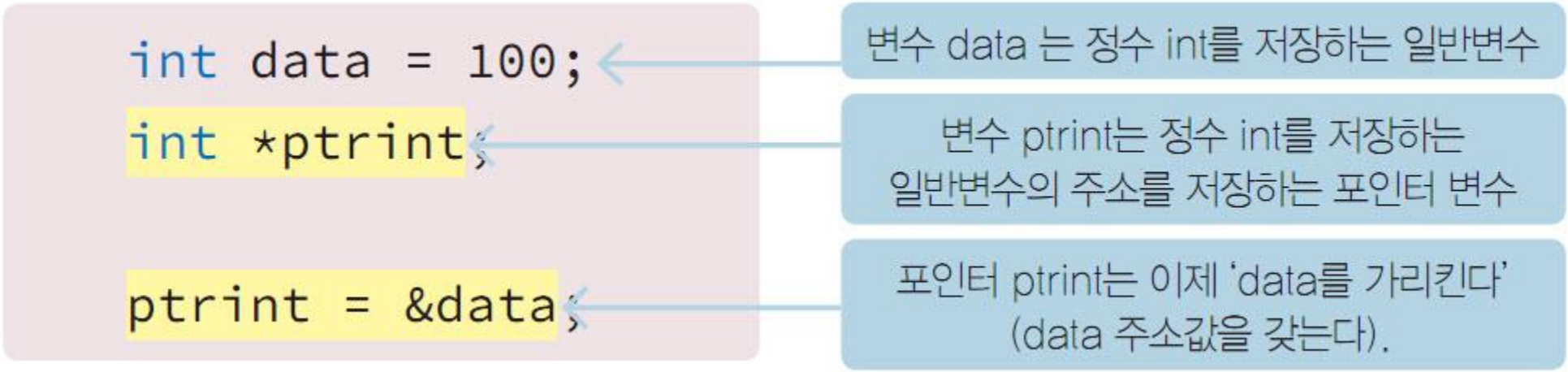
포인터 변수선언

자료형 *변수이름 ;

```
int *ptrint;
short *ptrshort;
char *ptrchar;
double *ptrdouble;
```

```
int *ptrint; //가장 선호
short*ptrshort;
char * ptrchar;
double *ptrdouble;
```

포인터 변수 선언


int data = 100;
int *ptrint;
ptrint = &data;
변수 data 는 정수 int를 저장하는 일반변수
변수 ptrint는 정수 int를 저장하는
일반변수의 주소를 저장하는 포인터 변수
포인터 ptrint는 이제 'data를 가리킨다'
(data 주소값을 갖는다).

Source Code #02: pointer.c

```c
// file: pointer.c
#include <stdio.h>

int main(void)
{
    int data = 100;
    int *ptrint;
    ptrint = &data;

    printf("변수명  주소값       저장값\n");
    printf("---------------------------\n");
    printf("  data  %p  %8d\n", &data, data);
    printf("ptrint  %p  %p\n", &ptrint, ptrint);

    return 0;
}
```

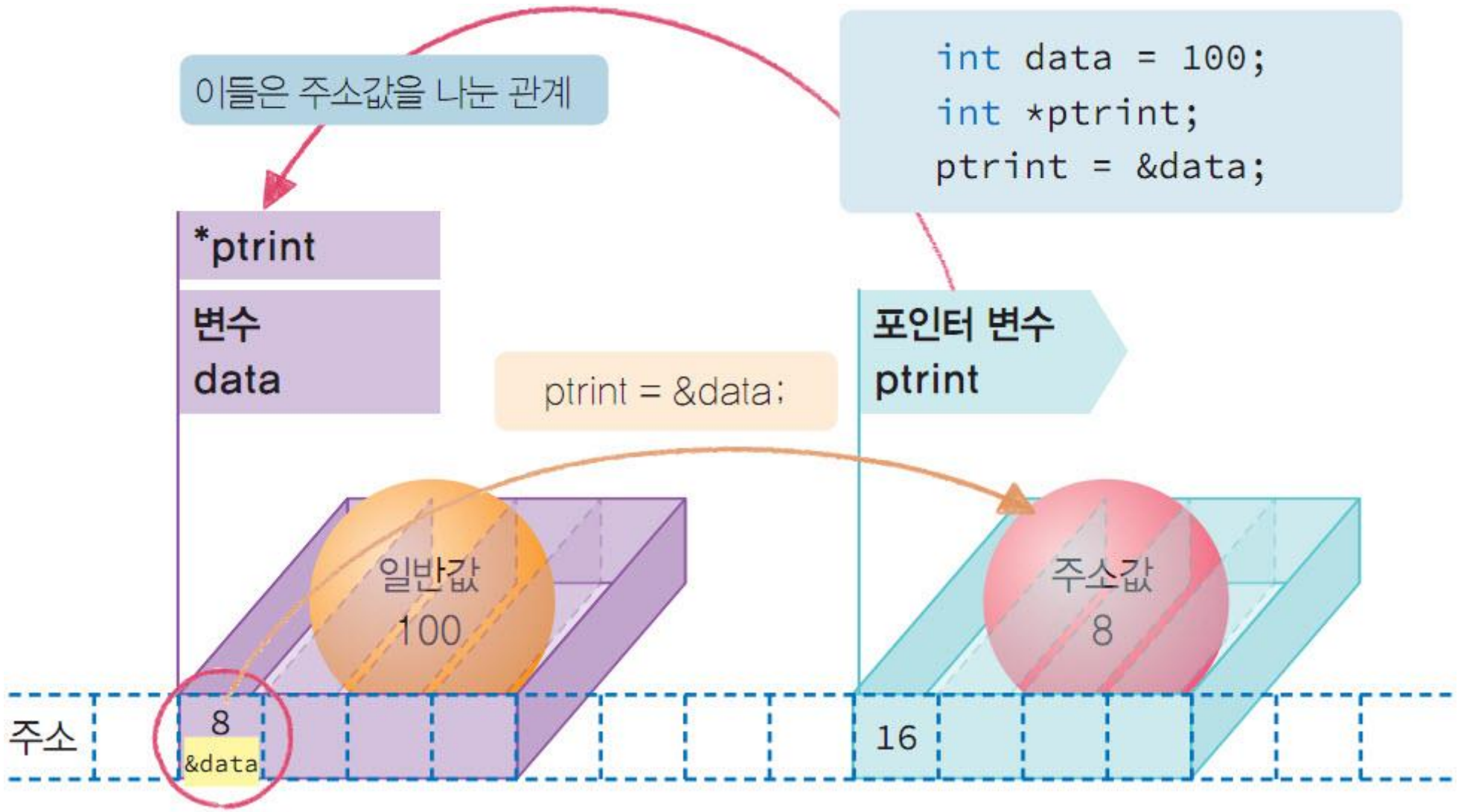
```
변수명  주소값       저장값
---------------------------
  data  00DEF9F8       100
ptrint  00DEF9EC  00DEF9F8
```

- 변수의 값과 주소값의 대입과 활용

포인터 선언과 대입

- 포인터 변수도 선언된 후 초기값이 없으면 의미 없는 쓰레기garbage 값이 저장
- 문장 ptrint = &data;
  - 포인터 변수 ptrint에 변수 data의 주소를 저장하는 문장
- &data에서 ptrint로의 화살표
  - 포인터 변수 ptrint에 변수 data의 주소가 저장되었다는 의미
  - 이러한 관계를 '포인터 변수 ptrint는 변수 data를 가리킨다' 또는 '참조reference한다'라고 표현
- 포인터 변수는 가리키는 변수의 크기
  - 종류에 관계없이 크기가 모두 4바이트

포인터 선언과 대입

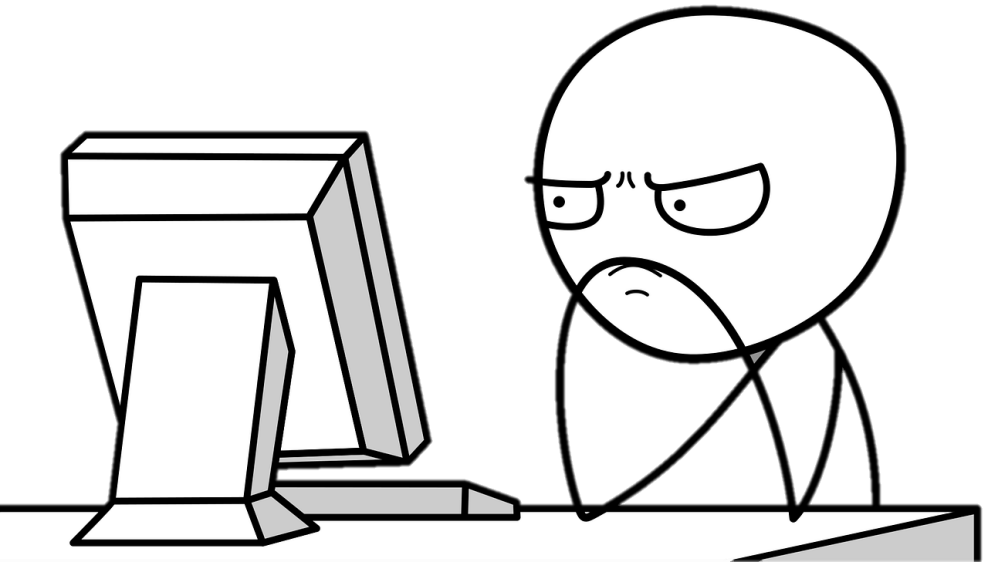

이들은 주소값을 나눈 관계
int data = 100;
int *ptrint;
ptrint = &data;
*ptrint
변수
data
ptrint = &data;
포인터 변수
ptrint
일반값
100
주소값
8
주소
8
&data
16

Lab #01: 다양한 자료형 포인터 변수 선언에 의한 주소값 출력

- 자료형 char, int, double의 변수와 포인터 변수 선언과 활용하는 프로그램
- char 포인터 변수선언: char *pc
- int 포인터 변수선언: int *pm
- double 포인터변수 선언: double *px

```c
// file: basicpointer.c
#include <stdio.h>

int main(void)
{
    char c = '@';
    char *pc = &c;
    int m = 100;
    int *pm = &m;
    double x = 5.83;
    double *px = &x;

    printf("변수명   주소값    저장값\n");
    printf("-------------------------\n");
    printf("%3s %12p %9c\n", "c", [illegible], c);
    printf("%3s %12p %9d\n", "m", [illegible], m);
    printf("%3s %12p %9f\n", "x", [illegible], x);

    return 0;
}
```

2. 간접 연산자 *와 포인터 연산



Source Code #03: nullpointer.c

```c
// file: nullpointer.c
#include <stdio.h>

int main(void)
{
    int *ptr1, *ptr2, data = 10;
    ptr1 = NULL;

    printf("%p\n", ptr1);
    //printf("%p\n", ptr2);
    printf("%d\n", data);

    return 0;
}
```

```
00000000
10
```

- 포인터 변수와 NULL의 활용
- 포인터 변수도 초기값을 저장하도록
  - 아니면 NULL을 저장
- 초기값을 지정하지 않은 포인터 변수 ptr2를 출력
  - "warning C4101: 'ptr2' : 참조되지 않은 지역 변수입니다."라는 컴파일오류가 발생

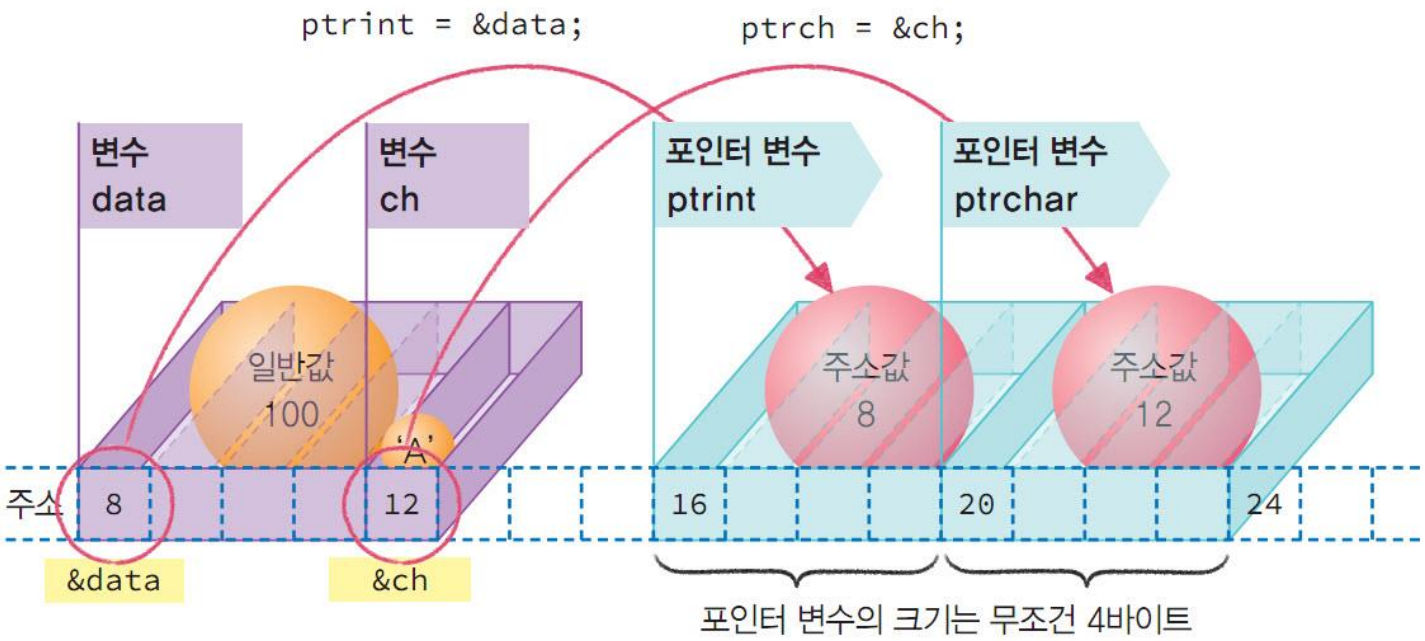
간접연산자 *

- 간접연산자indirection operator *를 사용
  - 포인터 변수가 갖는 주소로 그 주소의 원래 변수를 참조 가능
  - 포인터 변수가 가리키고 있는 변수를 참조
  - 간접연산자를 이용한 *ptrint
  - 포인터 ptrint가 가리키고 있는 변수 자체를 의미
- 직접참조direct access
  - 변수 data 자체를 사용해 자신을 참조하는 방식
- 간접참조indirect access
  - *ptrint를 이용해서 변수 data를 참조하는 방식

```
int data1 = 100, data2;
int *p;
printf("간접참조 출력: %d \n", *ptrint);

*ptrint = 200;
```

간접연산자 *

```
int data = 100;
int *ptrint = &data;
char *ptrchar = &ch;
printf("간접참조 출력: %d %c\n", *ptrint, *ptrchar);

*ptrint = 200;
printf("직접참조 출력: %d %c\n", data, ch);
```



Source Code #04: dereference.c

```c
// file: dereference.c
#include <stdio.h>

int main(void)
{
    int data = 100;
    char ch = 'A';
    int *ptrint = &data;
    char *ptrchar = &ch;
    printf("간접참조 출력: %d %c\n", *ptrint, *ptrchar);

    *ptrint = 200; //변수 data를 *ptrint로 간접참조하여 그 내용을 수정
    *ptrchar = 'B'; //변수 ch를 *ptrchar로 간접참조하여 그 내용을 수정
    printf("직접참조 출력: %d %c\n", data, ch);

    return 0;
}
```

- 포인터 변수와 간접연산자 *를 이용한 간접참조

```
간접참조 출력: 100 A
직접참조 출력: 200 B
```

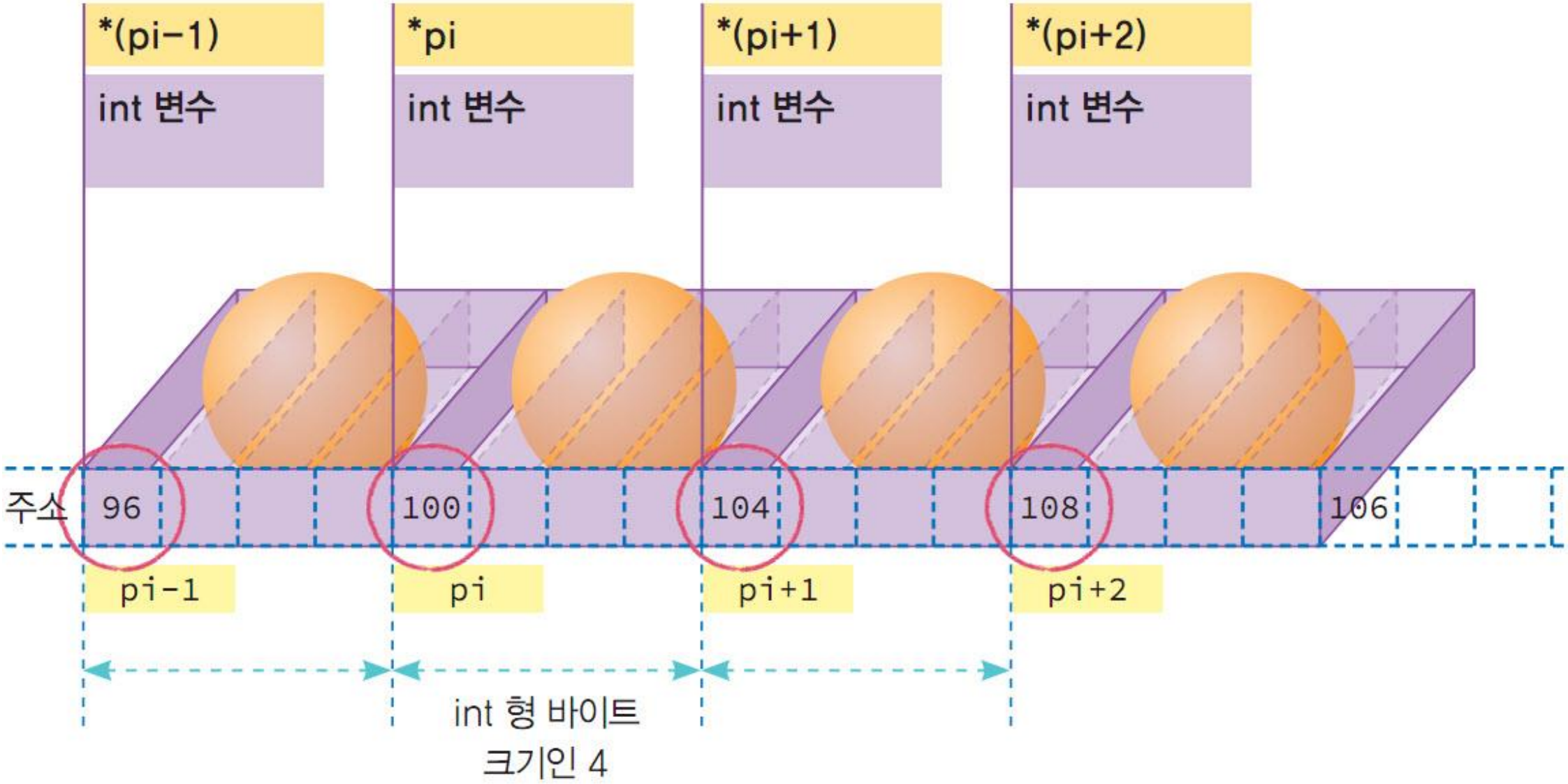
연산자 &와 *

- 주소 연산자 &와 간접 연산자 *, 모두 전위 연산자로 서로 반대의 역할
  - 주소 연산 '&변수'는 변수의 주소값이 결과값
  - 간접 연산 '*포인터변수'는 포인터 변수가 가리키는 변수 자체가 결과값
- '*포인터변수'는 l-value와 r-value로 모두 사용이 가능
  - 주소값인 '&변수'는 r-value로만 사용이 가능
- '*포인터변수'와 같이 간접연산자는 포인터 변수에만 사용이 가능
  - 주소 연산자는 '&변수'와 같이 모든 변수에 사용이 가능

주소 연산

- 포인터 변수는 간단한 더하기와 뺄셈 연산
  - 이웃한 변수의 주소 연산을 수행 가능
  - 포인터가 가리키는 변수 크기에 비례한 연산
  - 포인터의 연산은 절대적인 주소의 계산이 아님
  - 포인터에 저장된 주소값의 연산으로 이웃한 이전 또는 이후의 다른 변수를 참조
- int형 포인터 pi에 저장된 주소값이 100이라고 가정
  - (pi+1)은 101이 아니라 주소값 104
  - 즉 (pi+1)은 pi가 가리키는 다음 int형의 주소를 의미

주소 연산


*(pi-1)
int 변수
*pi
int 변수
*(pi+1)
int 변수
*(pi+2)
int 변수
주소
96
100
104
108
106
pi-1
pi
pi+1
pi+2
int 형 바이트
크기인 4

Source Code #05: calcptr.c

```c
// file: calcptr.c
#include <stdio.h>

int main(void)
{
    char *pc = (char *)100; //가능하나 잘 이용하지 않음
    int *pi = (int *)100;   //가능하나 잘 이용하지 않음
    double *pd = (double *)100;//가능하나 잘 이용하지 않음
    pd = 100;       //경고발생

    printf("%u  %u  %u\n", (int)(pc - 1), (int)pc, (int)(pc + 1));
    printf("%u  %u  %u\n", (int)(pi - 1), (int)pi, (int)(pi + 1));
    printf("%u  %u  %u\n", (int)(pd - 1), (int)pd, (int)(pd + 1));

    return 0;
}
```

```
99  100  101
96  100  104
92  100  108
```

- 다양한 자료형의 주소 연산과 주소 값 출력
- 포인터 pd에 정수 100을 직접 저장하면 경고가 발생
- double *pd = 100; //경고발생
- double *pd = (double *)100; //가능하나 잘 이용하지 않음
- double형 포인터에 100이라는 주소 값을 저장
- 포인터 자료형으로 100을 변환하는 연산식 (double *) 100을 사용해 저장 가능, 권장하지 않음

Source Code #06: neighborvar.c

```c
// file: neighborvar.c
#include <stdio.h>

int main(void)
{
    int a = 1, b = 3, c = 6;

    printf("변수명  저장값  주소값   \n");
    printf("----------------------\n");
    printf("   c     %d     %p\n", c, &c);
    printf("   b     %d     %p\n", b, &b);
    printf("   a     %d     %p\n", a, &a);

    int *p = &c;
    printf("   c     %d     %p\n", *p, p);
    printf("   b     %d     %p\n", *(p + 3), p + 3);
    printf("   a     %d     %p\n", *(p + 6), p + 6);

    return 0;
}
```

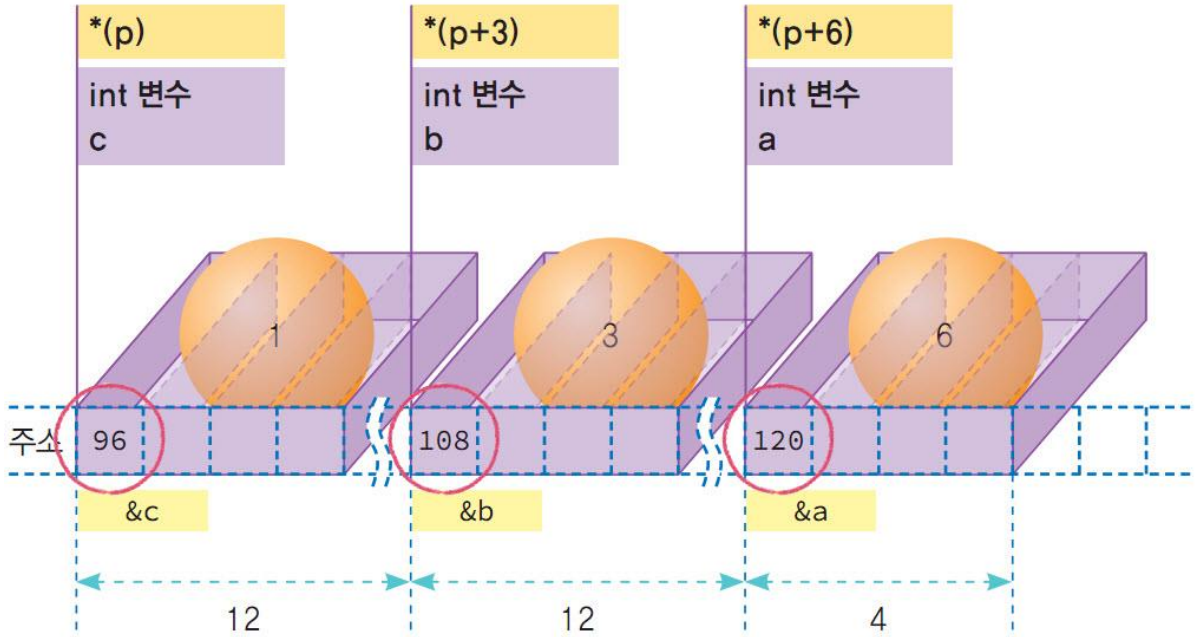
- int 자료형의 주소 연산과 주소값 출력

```
변수명  저장값  주소값
----------------------
   c     6     00EFF83C
   b     3     00EFF848
   a     1     00EFF854
   c     6     00EFF83C
   b     3     00EFF848
   a     1     00EFF854
```

이웃한 변수 주소

- int 형 변수 3개를 선언해 그 저장값과 주소값을 출력
- 일반적으로 정수 int와 int 사이는 주소값으로 그 차이가 절대 값으로 12 정도

```
int a = 1, b = 3, c = 6;
int *p = &c;

printf("   b     %d     %u\n", *(p + 3), p+3);
printf("   a     %d     %u\n", *(p + 6), p+6);
```

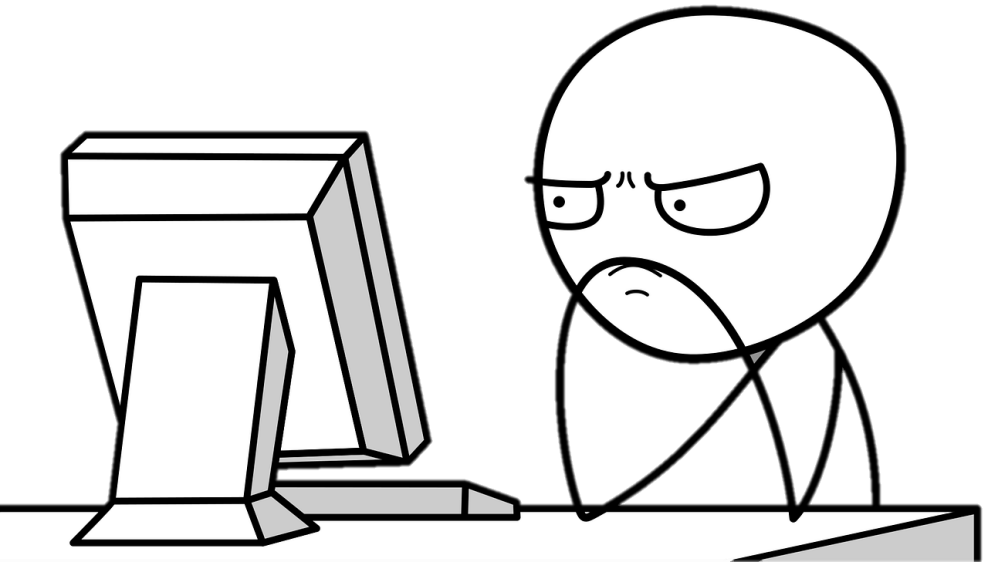


Lab #02: 포인터를 이용하여 두 수의 값을 교환하는 프로그램

- 정수 int 자료형 변수 m, n에 저장된 두 값을 서로 교환하는 프로그램
- 제한 사항
  - 임시변수인 dummy를 사용하고, 포인터 변수 p를 사용하나 변수 m, n 자체는 사용하지 않으며, 주소값 &m과 &n 만을 사용
  - 포인터 변수 선언 int *p = &m;으로 *p는 m 자체를 의미함
  - 마찬가지로 대입문장 p = &n;으로 *p는 n 자체를 의미함
- 결과
  - 100 200
  - 200 100

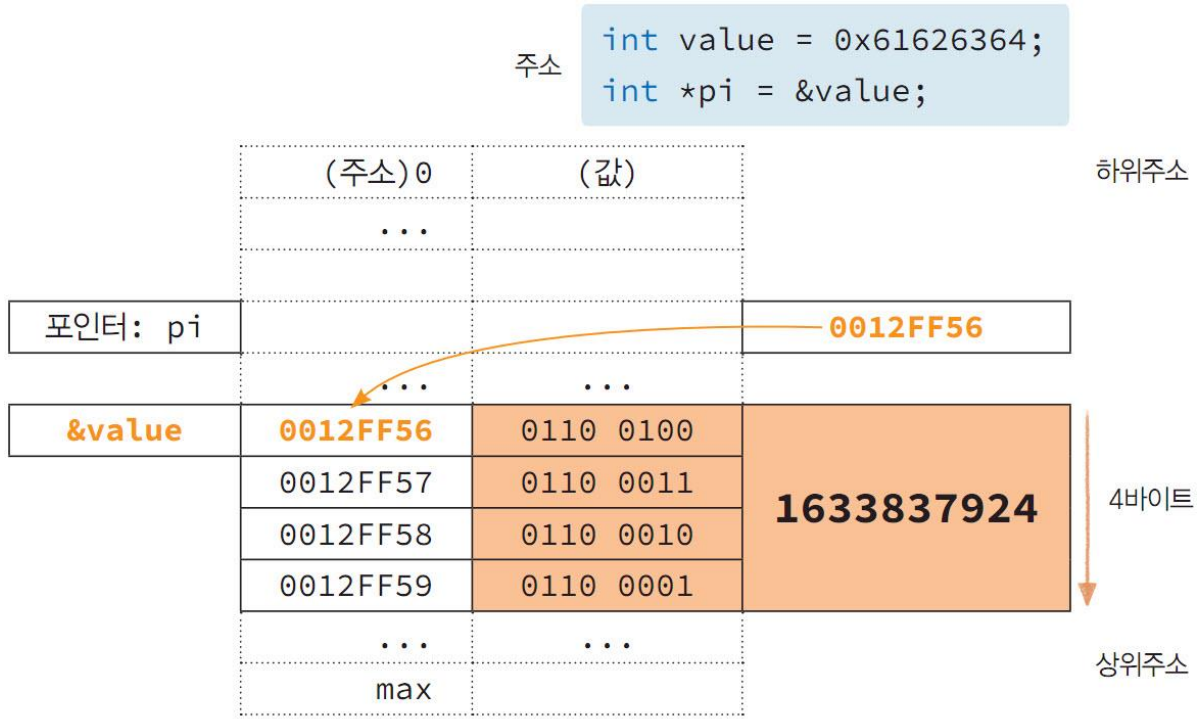
```c
// file: swap.c
#include <stdio.h>

int main(void)
{
    int m = 100, n = 200, dummy;
    printf("%d %d\n", m, n);

    //변수 m과 n을 사용하지 않고 두 변수를 서로 교환
    int *p = &m; //포인터 p가 m을 가리키도록
    [          ];  //변수 dummy에 m을 저장
    *p = n;       //변수 m에 n을 저장
    p = &n;       //포인터 p가 n을 가리키도록
    [          ];  //변수 n에 dummy 값 저장

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

    return 0;
}
```

3. 포인터 형변환과 다중 포인터



내부 저장 표현

- 변수 value에 16진수 0x61626364를 저장
  - 만일 변수 value의 주소가 56번지라면
  - 56번지에는 16진수 64가 저장
  - 다음 주소 57번지에는 63이 저장
  - 다음에 각각 62, 61이 저장



Source Code #07: ptrtypecast.c

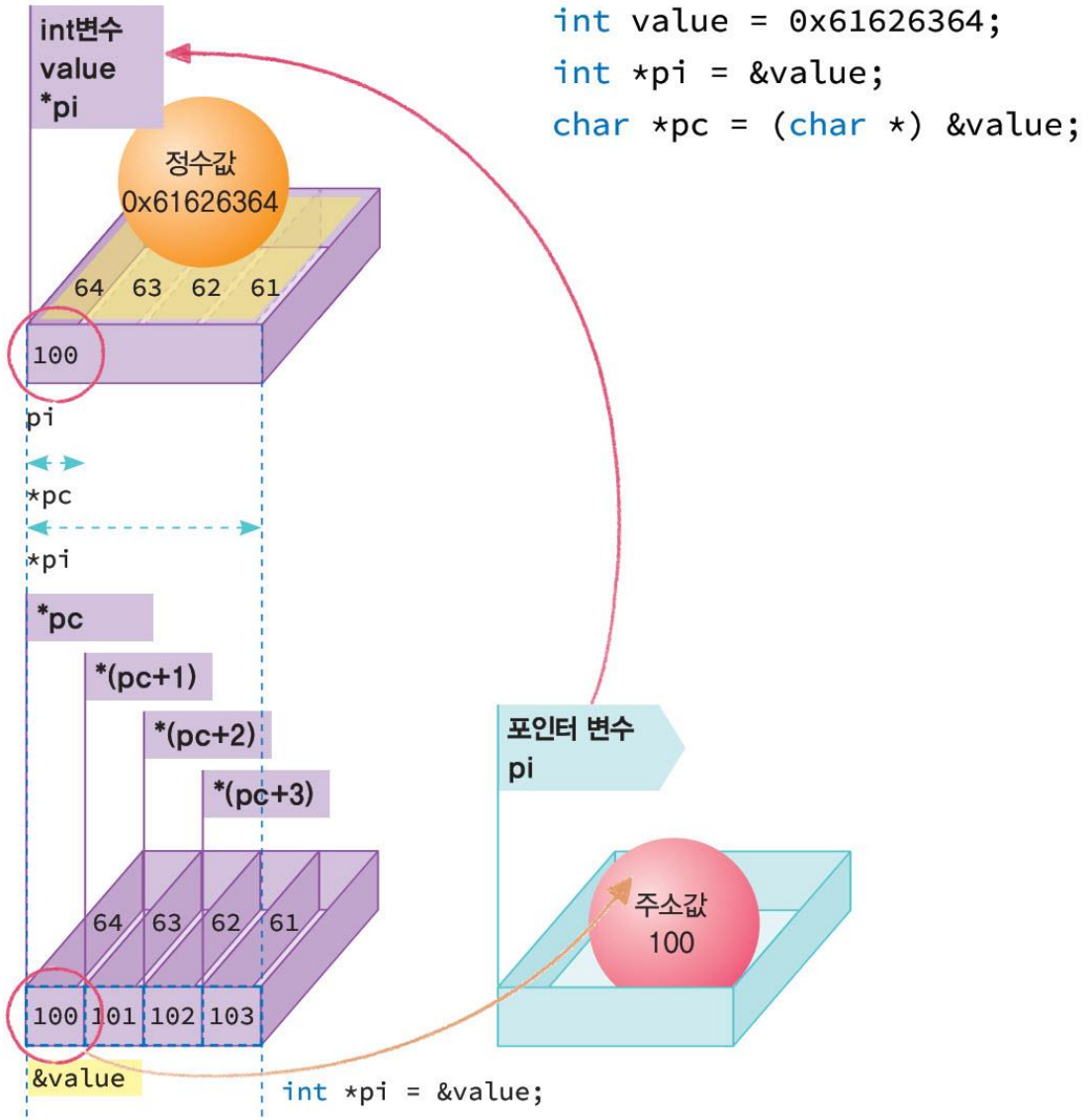
```c
// file: ptrtypecast.c
#include <stdio.h>

int main(void)
{
    int value = 0x61626364;
    int *pi = &value;
    char *pc = (char *)&value; //char *pc = &value;

    printf("변수명   저장값      주소값\n");
    printf("--------------------------\n");
    printf(" value   %0#x  %p\n", value, pi); //정수 출력

    //문자 포인터로 문자 출력 모듈
    for (int i = 0; i <= 3; i++)
    {
        char ch = *(pc + i);
        printf("*(pc+%d)  %0#6x  %2c  %p\n", i, ch, ch, pc + i);
    }

    return 0;
}
```

- 정수의 내부를 각각 1바이트 씩 문자로 출력

```
변수명   저장값      주소값
--------------------------
 value   0x61626364  00FCFD54
*(pc+0)  0x0064   d  00FCFD54
*(pc+1)  0x0063   c  00FCFD55
*(pc+2)  0x0062   b  00FCFD56
*(pc+3)  0x0061   a  00FCFD57
```

명시적 포인터 형변환

- 포인터 변수는 동일한 자료형끼리만 대입이 가능
- 포인터의 자료형이 다르면 경고가 발생
- 필요하면 명시적으로 형변환을 수행 가능

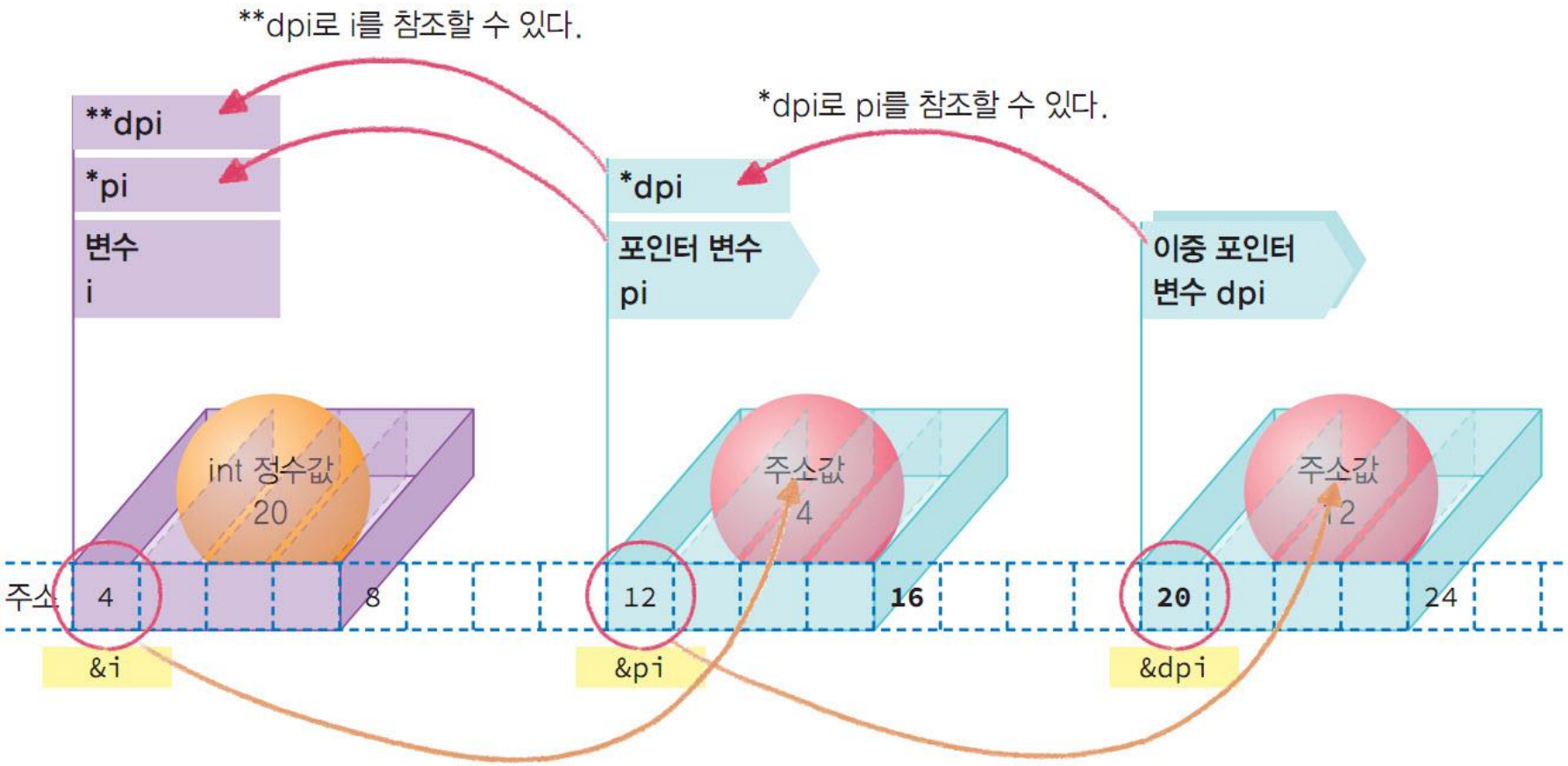


다중 포인터

- 이중 포인터
  - 포인터 변수의 주소값을 갖는 변수
- 삼중 포인터
  - 다시 이중 포인터의 주소값을 갖는 변수
- 다중 포인터
  - 포인터의 포인터

이중 포인터의 메모리와 변수

```
int i = 20;
int *pi = &i;
int **dpi = &pi;
```



Source Code #08: multipointer.c

```c
// file: multipointer.c
#include <stdio.h>

int main(void)
{
    int i = 20;
    int *pi = &i;       //포인터 선언
    int **dpi = &pi;    //이중 포인터 선언

    printf("%p %p %p\n", &i, pi, *dpi);

    *pi = i + 2;    // i = i + 2;
    printf("%d %d %d\n", i, *pi, **dpi);

    **dpi = *pi + 2;// i = i + 2;
    printf("%d %d %d\n", i, *pi, **dpi);

    return 0;
}
```

```
010FFBF4 010FFBF4 010FFBF4
22 22 22
24 24 24
```

- 포인터와 이중 포인터의 활용
- 다중 포인터 변수를 이용하여 일반 변수를 참조하려면 가리킨 횟수만큼 간접연산자를 이용
- 즉 이중 포인터 변수 dpi는 **dpi가 바로 변수 i
- 문장 *pi = i + 2;는 변수 i를 2 증가시키는 문장
  - 포인터 변수 pi에서 *pi도 변수 i
- 문장 **dpi = *pi + 2;는 변수 i를 2 증가시키는 문장

간접 연산자와 증감 연산자 활용

- 간접 연산자 *는 증감 연산자 ++, --와 함께 사용하는 경우
  - 간접 연산자 *는 전위 연산자로 연산자 우선순위가 2위
  - 증감 연산자 ++, --는 전위이면 2위이고, 후위이면 1위

우선순위	단항 연산자	설명	결합성(계산방향)
1	a++ a--	후위 증가, 후위 감소	-> (좌에서 우로)
2	++a --a & *	전위 증가, 전위 감소 주소 간접 또는 역참조	<- (우에서 좌로)

간접 연산자와 증감 연산자 활용

- 사용 사례
  - *p++는 *(p++)으로 (*p)++와 다르다.
  - ++*p와 ++(*p)는 같다.
  - *++p는 *(++p)는 같다.

연산식		결과값	연산 후 *p의 값	연산 후 p 증가
*p++	*(p++)	*p: p의 간접참조 값	변동 없음	p+1: p 다음 주소
*++p	*(++p)	*(p+1): p 다음 주소 (p+1) 간접참조 값	변동 없음	p+1: p 다음 주소
(*p)++		*p: p의 간접참조 값	*p가 1 증가	p: 없음
++*p	++(*p)	*p + 1: p의 간접참조 값에 1 증가	*p가 1 증가	p: 없음

Source Code #09: variousop.c

```c
// file: variousop.c
#include <stdio.h>

int main(void)
{
    int i;
    int *pi = &i;       //포인터 선언
    int **dpi = &pi;    //이중포인터 선언

    *pi = 5;
    *pi += 1; //*pi = *pi + 1와 같음
    printf("%d\n", i);

    // 후위 연산자 pi++는 전위 연산자보다 *pi보다 빠름
    printf("%d\n", (*pi)++); //*pi++ 는 *(pi++)으로 (*pi)++과 다름
    printf("%d\n", *pi);

    *pi = 10;
    printf("%d\n", ++*pi); //++*pi과 ++(*pi)는 같음
    printf("%d\n", ++**dpi); //++**dpi과 ++(**dpi)는 같음
    printf("%d\n", i);

    return 0;
}
```

- 간접연산자와 증가연산자의 활용

```
6
6
7
11
12
12
```

Source Code #10: constptr.c

```c
/* constptr.c */
#include <stdio.h>

int main()
{
    int i = 10, j = 20;
    const int *p = &i; //*p가 상수로 *p로 수정할 수 없음
    //*p = 20; //오류 발생
    p = &j;
    printf("%d\n", *p);

    double d = 7.8, e = 2.7;
    double * const pd = &d;
    //pd = &e; //pd가 상수로 다른 주소 값을 저장할 수 없음
    *pd = 4.4;
    printf("%f\n", *pd);

    return 0;
}
```

- 포인터 상수 활용

```
20
4.400000
```

포인터 상수

- *pi를 사용해 포인터 pi가 가리키는 변수인 i를 수정할 수 없도록 하는 상수 선언 방법
  - 즉 간접 연산식 *pi을 상수로 만들면 *pi를 l-value로 사용할 수 없음
- 1번과 동일한 문장으로 간접 연산식 *pi를 상수로 만드는 방법
- 포인터 pi에 저장되는 초기 주소값을 더 이상 수정할 수 없도록 하는 상수 선언 방법
  - 즉 포인터 변수 pi 자체를 상수로 만드는 방법으로, 선언 이후 pi를 l-value로 사용할 수 없음

```
int i = 10, j = 20;
```

❶
```
const int *pi = &i;
```
```
*pi = 20; //오류 발생
```
const int *pi
*pi가 상수로 *pi로 수정할 수 없음
오류: 식이 수정할 수 있는 lvalue여야 합니다.

❷
```
int const *pi = &i;
```
```
*pi = 20; //오류 발생
```
const int *pi
*pi가 상수로 *pi로 수정할 수 없음
오류: 식이 수정할 수 있는 lvalue여야 합니다.

❸
```
int* const pi = &i;
```
```
pi = &j; //오류 발생
```
int *const pi
오류: 식이 수정할 수 있는 lvalue여야 합니다.

Lab #03: 두 실수의 덧셈을 포인터 변수를 사용해 수행하고 출력

- 자료형 double로 선언된 두 x와 y에 표준입력으로 두 실수를 입력 받아 두 실수의 덧셈 결과를 출력하는 프로그램
- 제한 사항
  - 두 변수 x와 y는 선언만 수행하며, 포인터 변수인 px와 py만을 사용하여 모든 과정을 코딩
  - double 포인터 변수 px 선언:
    double *px = &x;
  - double 포인터 변수 py 선언:
    double *py = &y;
- 결과
  - 두 실수 입력: 3.874 7.983
  - 3.87 + 7.98 = 11.86

```c
// file: sumpointer.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    double x, y;
    double *px = &x;
    double *py = &y;

    //포인터 변수 px와 py를 사용
    printf("두 실수 입력: ");
    scanf("%lf %lf", [redacted]);
    //합 출력
    printf("%.2f + %.2f = %.2f\n", [redacted]);

    return 0;
}
```


I
PROGRAMMING