제08장

포인터 기초

Perfect C

C언어로 배우는 프로그래밍 기초

단원 목표

학습목표

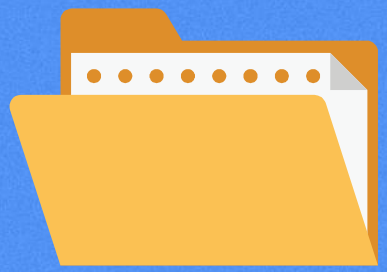
- **포인터 변수를 이해하고 설명할 수 있다.**
  - 메모리와 주소, 주소 연산자 &
  - *를 사용한 포인터 변수 선언과 간접참조 방법
  - 포인터 변수의 연산과 형변환
- **다중 포인터와 배열 포인터를 이해하고 설명할 수 있다.**
  - 이중 포인터의 필요성과 선언 및 사용 방법
  - 증감연산자와 포인터와의 표현식
  - 포인터 상수

학습목차

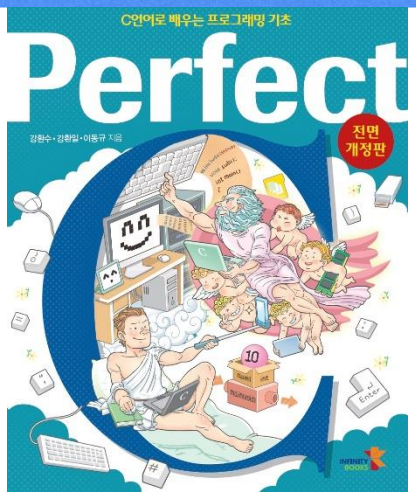
8.1 포인터 변수와 선언

8.2 간접 연산자 *와 포인터 연산

8.3 포인터 형변환과 다중 포인터



01. 포인터 변수와 선언



주소 개념

- **주소(address)**
  - 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

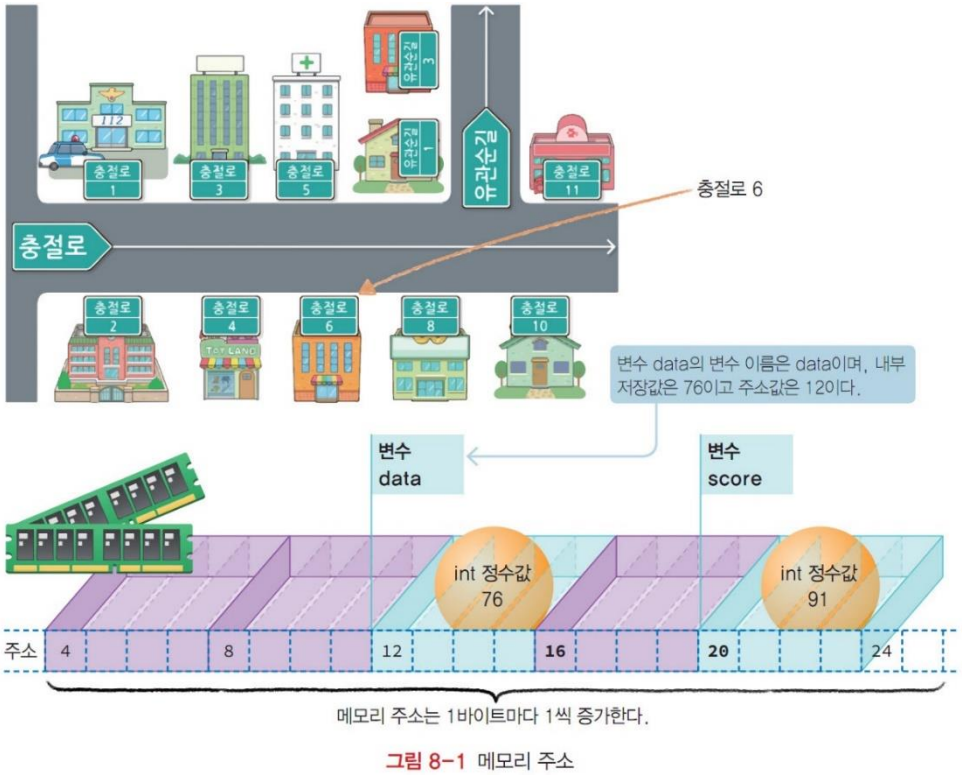


그림 8-1 메모리 주소

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

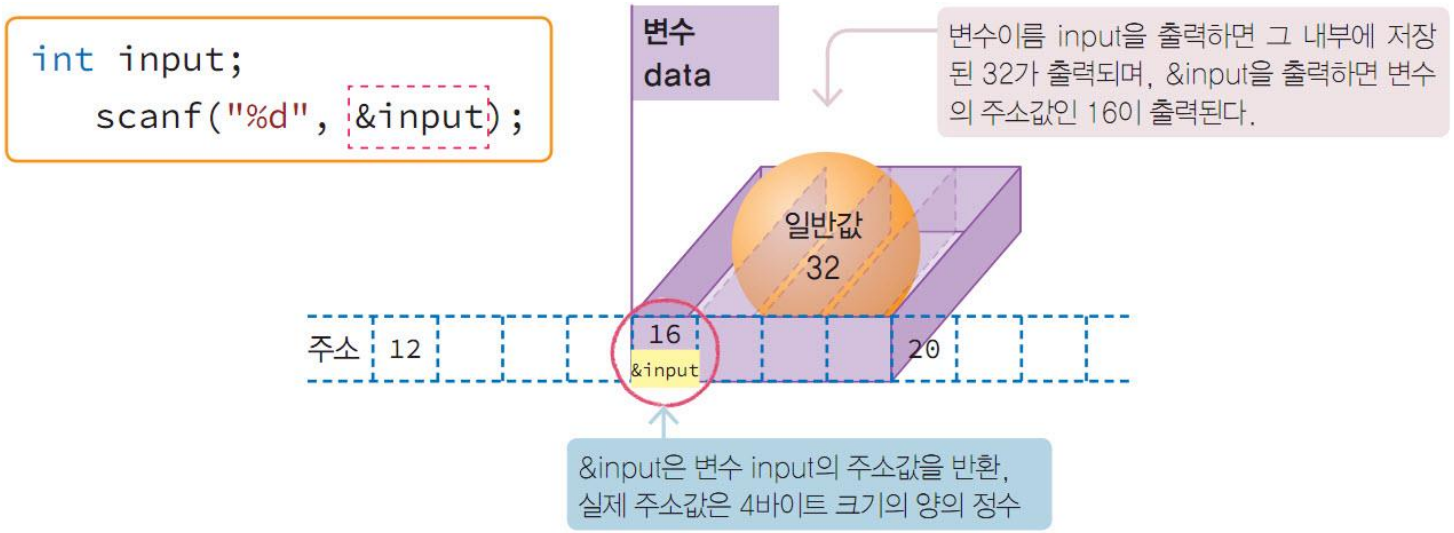


그림 8-2 주소연산자 & 이해

주소값 출력

예제 address.c

- 변수의 값과 주소 값을 출력

제어변수

- 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

실습예제 8-1 address.c

```
01  // file: address.c
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04
05  int main(void)
06  {
07      int input;
08
09      printf("정수 입력: ");
10      scanf("%d", &input);
11      printf("입력값: %d\n", input);
12      printf("주소값: %u(10진수), %p(16진수)\n", (int) &input, &input);
13      printf("주소값: %d(10진수), %#X(16진수)\n", (unsigned) &input,
            (int) &input);
14      printf("주소값 크기: %d\n", sizeof(&input));
15
16      return 0;
17  }
```

설명

```
07  int 형 변수 input 선언
09  저장값 입력을 위한 프롬프트
10  표준입력에서 반드시 &input
12  변수 data의 주소값을 참조하려면 &input을 사용하며, 표준출력에서 %u, %p등 사용
13  변수 data의 주소값을 참조하려면 &input을 사용하며, 표준출력에서 %d, %#X등 사용,
    변환명세에서 필요한 자료값으로 변환
14  주소값의 크기도 4바이트
```

실행결과

```
정수 입력: 100
입력값: 100
주소값: 3799464(10진수), 0039F9A8(16진수)
주소값: 3799464(10진수), 0X39F9A8(16진수)
주소값 크기: 4
```

%p

%#X

메모리 주소 저장 변수인 포인터 변수

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

그림 8-3 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

포인터 변수선언

자료형 *변수이름 ;

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

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

그림 8-4 포인터 변수선언 구문

  - 예로 'int *ptrint' 선언
    - 'int 포인터 ptrint'라고 읽도록
  - 변수 자료형이 다르면
    - 그 변수의 주소를 저장하는 포인터의 자료형도 반드시 달라야 함

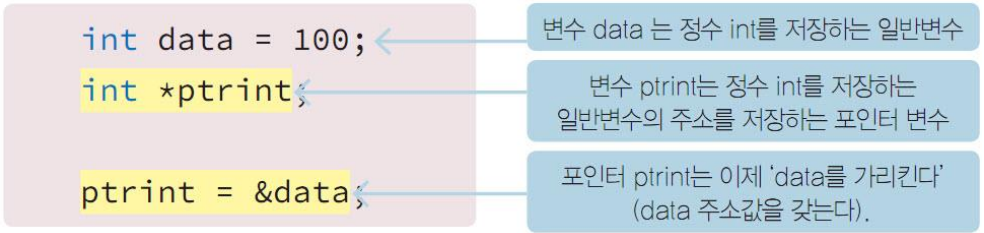


그림 8-5 포인터 변수와 일반 변수의 주소 저장

포인터 선언과 대입

예제 pointer.c

- 변수의 값과 주소 값의 대입과 활용

제어변수

- 포인터 변수도 선언된 후 초기값이 없으면
  - 의미 없는 쓰레기(garbage) 값이 저장
- 문장 ptrint = &data;
  - 포인터 변수 ptrint에 변수 data의 주소를 저장하는 문장

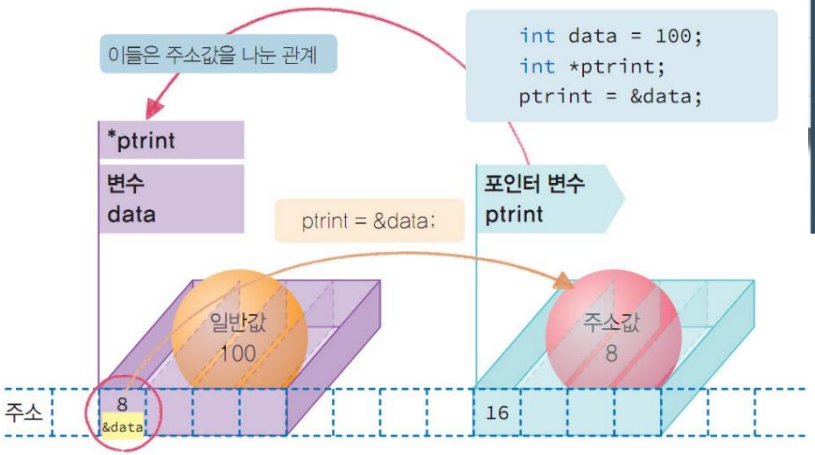
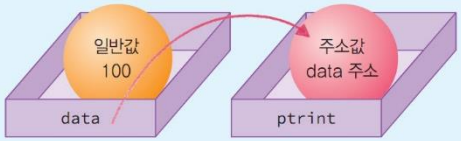


그림 8-6 포인터 변수선언과 주소의 대입

실습예제 8-2 pointer.c

포인터 변수선언과 주소값 대입

```
01  // file: pointer.c
02  #include <stdio.h>
03
04  int main(void)
05  {
06     int data = 100;
07     int *ptrint;
08     ptrint = &data;
09
10     printf("변수명  주소값        저장값\n");
11     printf("----------------------------\n");
12     printf("  data  %p  %8d\n", &data, data);
13     printf("ptrint  %p  %p\n", &ptrint, ptrint);
14
15     return 0;
16  }
```



설명

```
06  int 형 변수 data 선언하여 100 저장
07  int 형 포인터 변수 ptrint 선언
08  int 형 포인터 변수 ptrint에 변수 data의 주소를 저장
12  변수 data의 주소(0024FB44)와 내부 저장값 100을 출력
13  변수 ptrint의 주소(0024FB38)와 내부 저장값 0024FB44를 출력, 변수 ptrint의 내부 저장
    값은 바로 data의 주소값인 0024FB44인 것을 확인할 수 있으며, 이러한 주소값은 실행 때마다
    다를 수 있음
```

실행결과

```
변수명      주소값       저장값
----------------------------
  data  0024FB44        100
ptrint  0024FB38  0024FB44
```

- &data에서 ptrint로의 화살표
  - 포인터 변수 ptrint에 변수 data의 주소가 저장되었다는 의미
  - 이러한 관계를 '포인터 변수 ptrint는 변수 data를 가리킨다' 또는 '참조(reference)한다'라고 표현
- 포인터 변수는 가리키는 변수의 크기
  - 종류에 관계없이 크기가 모두 4바이트

LAB 다양한 자료형 포인터 변수 선언에 의한 주소값 출

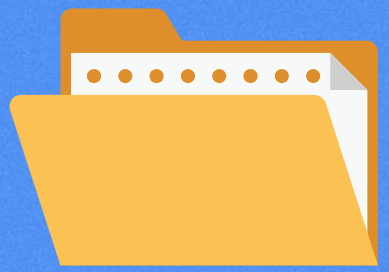
- **자료형 char, int, double의 변수와 포인터 변수 선언과 활용하는 프로그램**
  - char 포인터 변수 선언: char *pc
  - int 포인터 변수 선언: int *pm
  - double 포인터 변수 선언: double *px

Lab 8-1 basicpointer.c

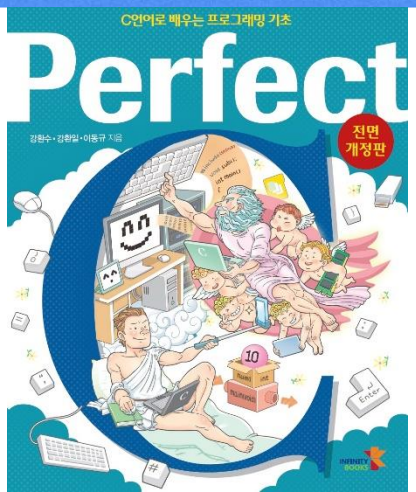
```c
// file: basicpointer.c
#include <stdio.h>

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

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

   return 0;
}
```

정답

```c
07  char *pc = &c;
09  int *pm = &m;
11  double *px = &x;
15  printf("%3s %12p %9c\n", "c", pc, c);
16  printf("%3s %12p %9d\n", "m", pm, m);
17  printf("%3s %12p %9f\n", "x", px, x);
```



02. 간접 연산자 *와 포인터 연산



여러 포인터 변수선언

- **여러 개의 포인터 변수를 한 번에 선언**
  - 두 번째 선언은 ptr1은 int 형 포인터를 의미하나 ptr2와 ptr3은 단순히 int형
- **포인터 변수에 지정할 특별한 초기값이 없는 경우**
  - int *ptr = NULL;
    - 0번 주소값인 NULL로 초기값을 저장
    - NULL이 저장된 포인터 변수는 아무 변수도 가리키고 있지 않다는 의미
- **NULL**
  - 헤더파일 stdio.h에 정의되어 있는 포인터 상수로서 0번지의 주소값을 의미
    - #define NULL ((void *)0)
  - 여기서 (void *)는 아직 결정되지 않은 자료형의 주소를 의미
  - 자료유형 (void *)는 아직 유보된 포인터이므로 모든 유형의 포인터 값을 저장할 수 있는 포인터 형

주소값 NULL

예제 nullpointer.c

- 포인터 변수와 NULL의 활용

제어변수

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

실습예제 8-3 nullpointer.c

한번에 여러 포인터 변수 선언과 NULL 주소값 대입

```
01  // file: nullpointer.c
02  #include <stdio.h>
03
04  int main(void)
05  {
06     int *ptr1, *ptr2, data = 10;
07     ptr1 = NULL;
08
09     printf("%p\n", ptr1);
10     //printf("%p\n", ptr2);   <- 초기값이 없어서 컴파일 오류발생
11     printf("%d\n", data);
12
13     return 0;
14  }
```

설명

```
06  int 형 포인터 변수 ptr1과 ptr2를 선언하면서 int 형 변수 data 선언하여 10 저장
07  int 형 포인터 변수 ptr1에 주소값 0인 NULL을 저장
09  int 형 포인터 변수 ptr1에 저장된 주소값인 0을 출력
10  int 형 포인터 변수 ptr2에 저장된 주소값을 출력하려 하나 저장된 것이 없어 컴파일오류 발생
11  int 형 변수 data에 저장된 값인 10을 출력
```

실행결과

```
00000000
10
```

간접연산자 *

- **간접연산자(indirection 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
- **직접참조(direct access)**
  - 변수 data 자체를 사용해 자신을 참조하는 방식
- **간접참조(indirect access)**
  - *ptrint를 이용해서 변수 data를 참조하는 방식

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

*ptrint = 200;
```

그림 8-8 간접연산자 *와 간접참조

```
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);
```

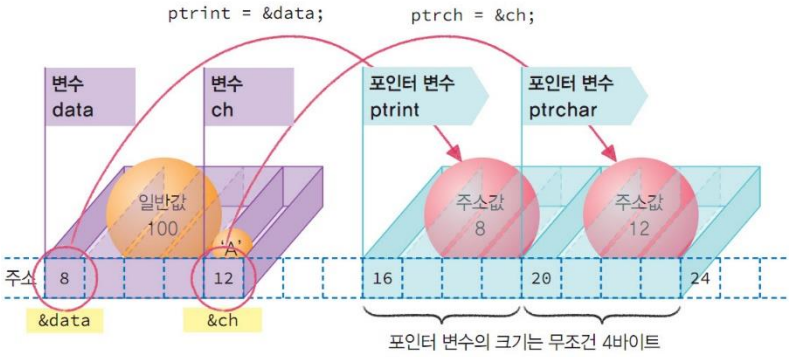


그림 8-9 포인터 변수와 주소값이 저장된 변수 사이의 관계

간접참조

예제 dereference.c

- 포인터 변수와 간접연산자 *를 이용한 간접참조

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

실습예제 8-4 dereference.c

포인터 변수와 간접연산자 *를 이용한 간접참조

```
01  // file: dereference.c
02  #include <stdio.h>
03
04  int main(void)
05  {
06     int data = 100;
07     char ch = 'A';
08     int *ptrint = &data;
09     char *ptrchar = &ch;
10     printf("간접참조 출력: %d %c\n", *ptrint, *ptrchar);
11
12     *ptrint = 200; //변수 data를 *ptrint로 간접참조하여 그 내용을 수정
13     *ptrchar = 'B'; //변수 ch를 *ptrchar로 간접참조하여 그 내용을 수정
14     printf("직접참조 출력: %d %c\n", data, ch);
15
16     return 0;
17  }
```

변수 data와 같다. / 변수 ch와 같다.

설명

```
06  int 형 변수 data 선언하여 100 저장
07  char 형 변수 ch 선언하여 문자 'A' 저장
08  int 형 포인터 변수 ptrint를 선언하여 변수 data의 주소값을 저장
09  char 형 포인터 변수 ptrchar를 선언하여 변수 ch의 주소값을 저장
10  int 형 포인터 변수 ptrint와 간접연산자 *를 이용한 *ptrint를 사용하여 data의 저장값을 출력
10  char 형 포인터 변수 ptrchar와 간접연산자 *를 이용한 *ptrchar를 사용하여 ch의 저장값을
    출력
12  int 형 포인터 변수 ptrint와 간접연산자 *를 이용한 *ptrint를 사용하여 data의 저장값을
    200으로 수정
13  char 형 포인터 변수 ptrchar와 간접연산자 *를 이용한 *ptrchar를 사용하여 ch의 저장값을
    'B'로 수정
14  int 형 변수 data에 저장된 값과 char 형 변수 ch에 저장된 값을 출력
```

실행결과

```
간접참조 출력: 100 A
직접참조 출력: 200 B
```

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 아니며
    - 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

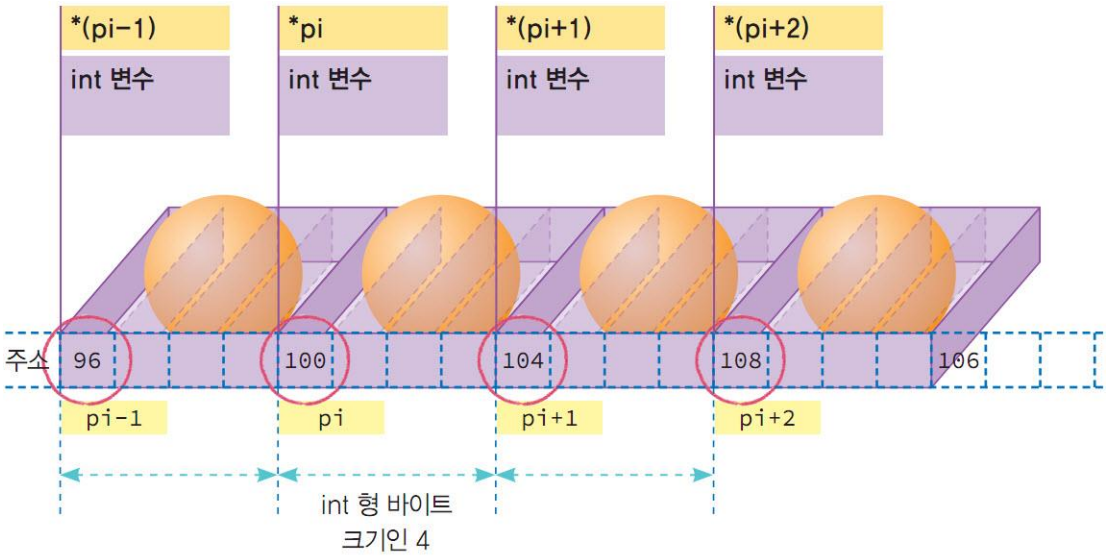


그림 8-10 주소의 연산을 이용한 이웃한 변수의 참조

주소 연산

예제 calcptr.c

- 다양한 자료형의 주소 연산과 주소 값 출력

연산자 &와 *

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

실습예제 8-5 calcptr.c

포인터 변수의 간단한 덧셈 뺄셈 연산

```
01  // file: calcptr.c
02  #include <stdio.h>
03
04  int main(void)
05  {
06     char *pc = (char *)100;//가능하나 잘 이용하지 않음
07     int *pi = (int *)100;   //가능하나 잘 이용하지 않음
08     double *pd = (double *)100;//가능하나 잘 이용하지 않음
09     pd = 100;       //경고발생
10
11     printf("%u  %u  %u\n", (int)(pc - 1), (int)pc, (int)(pc + 1));
12     printf("%u  %u  %u\n", (int)(pi - 1), (int)pi, (int)(pi + 1));
13     printf("%u  %u  %u\n", (int)(pd - 1), (int)pd, (int)(pd + 1));
14
15     return 0;
16  }
```

설명

```
06  char형 포인터 변수 pc 선언하여 char형 포인터 주소값 100을 저장
07  int형 포인터 변수 pi 선언하여 int형 포인터 주소값 100을 저장
08  double형 포인터 변수 pd 선언하여 double형 포인터 주소값 100을 저장
09  double형 포인터 변수 pd에 정수 100을 저장하므로 경고 발생
11  char형 포인터 변수 pc에 저장된 값 100과 비교하여 pc-1과 pc+1을 출력, char의 크기가
    1이므로 각각 1만큼의 차이가 나므로 99 100 101 출력
12  int형 포인터 변수 pi에 저장된 값 100과 비교하여 pi-1과 pi+1을 출력, int의 크기가
    4이므로 각각 4만큼의 차이가 나므로 96 100 104 출력
13  double형 포인터 변수 pd에 저장된 값 100과 비교하여 pd-1과 pd+1을 출력, double의 크기가
    8이므로 각각 8만큼의 차이가 나므로 92 100 108 출력
```

실행결과

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

이웃한 변수 주소

예제 neighborvar.c

- int 자료형의 주소 연산과 주소 값 출력

연산자 &와 *

- 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);
```

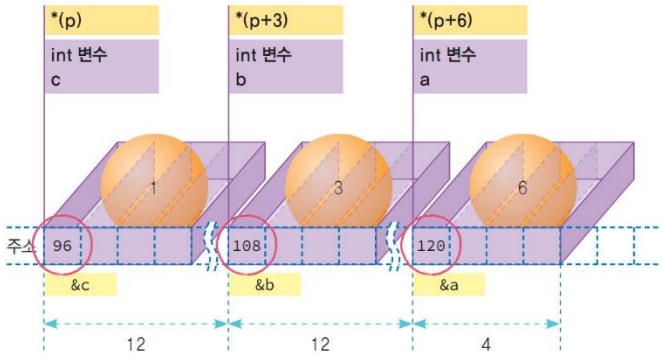


그림 8-11 포인터 변수를 이용한 배열의 참조

실습예제 8-6 neighborvar.c

간접연산자와 증감연산자의 이용

```
01 // file: neighborvar.c
02 #include <stdio.h>
03
04 int main(void)
05 {
06    int a = 1, b = 3, c = 6;
07
08    printf("변수명  저장값  주소값   \n");
09    printf("------------------------\n");
10    printf("   c     %d     %p\n", c, &c);
11    printf("   b     %d     %p\n", b, &b);
12    printf("   a     %d     %p\n", a, &a);
13
14    int *p = &c;
15    printf("   c     %d     %p\n", *p, p);
16    printf("   b     %d     %p\n", *(p + 3), p+3);
17    printf("   a     %d     %p\n", *(p + 6), p+6);
18
19    return 0;
20 }
```

설명

```
01 int 형 변수 a, b, c를 선언하여 각각 1, 3, 6을 저장
08 출력을 위한 제목 헤드라인 출력
09 출력을 위한 헤드라인 출력
10 변수 c의 변수명 저장값 주소값 출력
11 변수 b의 변수명 저장값 주소값 출력, 변수 b와 c의 주소 차이는 12
12 변수 a의 변수명 저장값 주소값 출력, 변수 a와 b의 주소 차이는 12
14 int 형 포인터 변수 p 선언하여 int 형 변수 c의 주소값 &c를 저장
15 변수 c의 변수명 저장값 주소값 출력, 저장값은 *p로, 주소값은 p로 출력
16 변수 b의 변수명 저장값 주소값 출력, 저장값은 *(p + 3)으로, 주소값은 p+3으로 출력, int는
   하나의 차이가 4이므로 12는 p+3으로 이동
17 변수 a의 변수명 저장값 주소값 출력, 저장값은 *(p + 6)으로, 주소값은 p+6으로 출력, int는
   하나의 차이가 4이므로 24는 p+6으로 이동
```

실행결과

```
변수명  저장값      주소값
------------------------
   c     6     0033FCC0
   b     3     0033FCCC
   a     1     0033FCD8
   c     6     0033FCC0
   b     3     0033FCCC
   a     1     0033FCD8
```

LAB 포인터를 이용하여 두 수의 값을 교환하는 프로그램

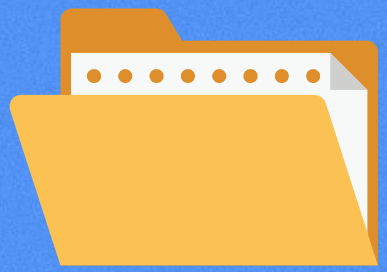
- **정수 int 자료형 변수 m, n에 저장된 두 값을 서로 교환하는 프로그램**
- **제한 사항**
  - 임시변수인 dummy를 사용하고, 포인터 변수 p를 사용하나 변수 m, n 자체는 사용하지 않으며, 주소값 &m과 &n 만을 사용
  - 포인터 변수 선언 int *p = &m;으로 *p는 m 자체를 의미함
  - 마찬가지로 대입문장 p = &n;으로 *p는 n 자체를 의미함
- **결과**
  - 100 200
  - 200 100

Lab 8-2 swap.c

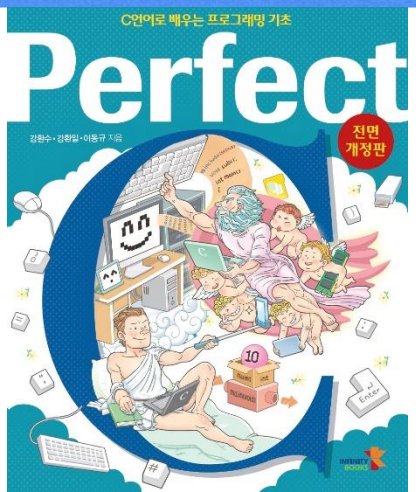
```c
// file: swap.c
#include <stdio.h>

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;
}
```

정답

```c
dummy = *p;  //변수 dummy에 m을 저장
*p = dummy;  //변수 n에 dummy 값 저장
```



03. 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

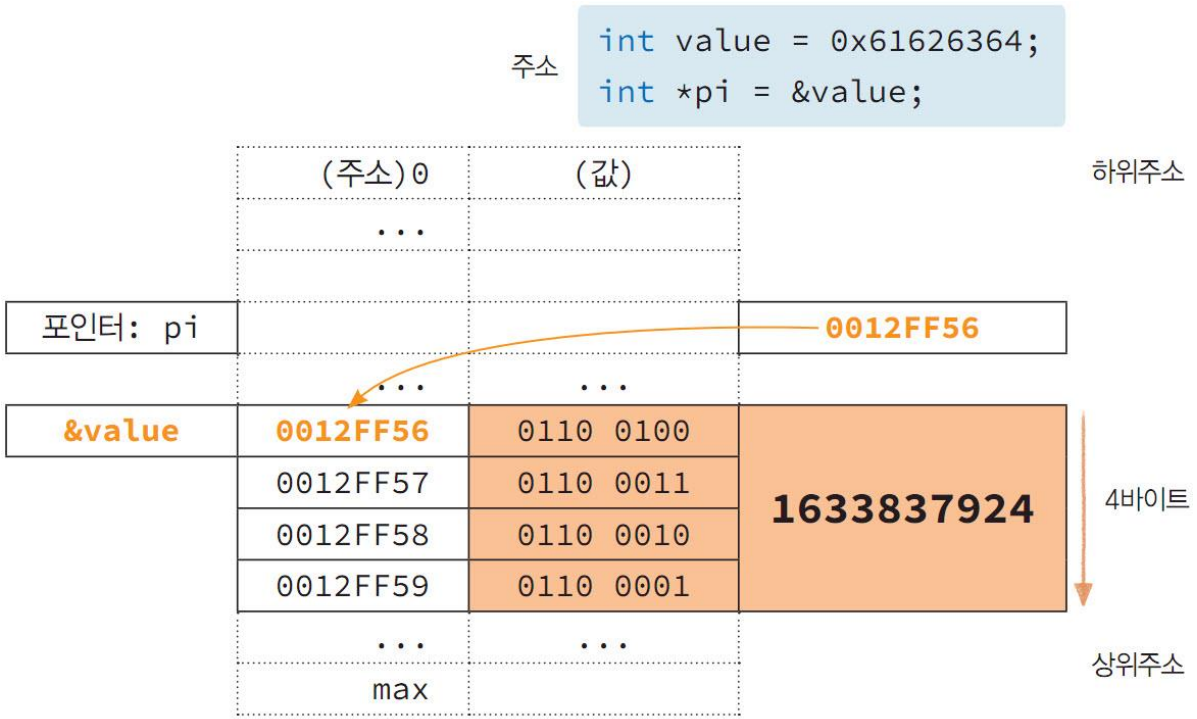


그림 8-12 지역변수가 선언된 메모리 내부 모습

명시적 포인터 형변환

예제 ptrtypecast.c

- 정수의 내부를 각각 1바이트 씩 문자로 출력

포인터 형변환

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

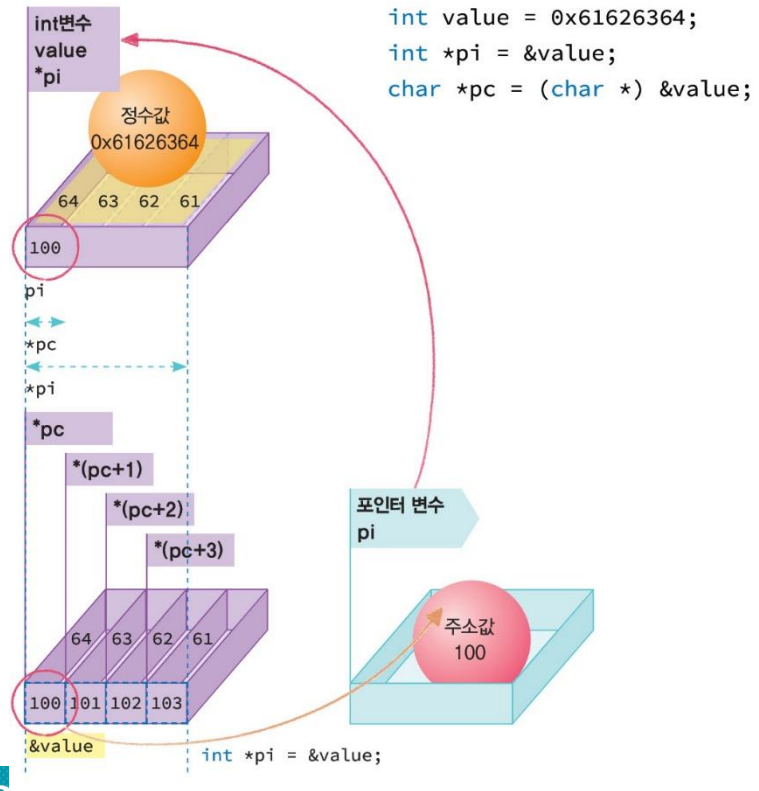


그림 8-14 동일한 메모리의 내용이 포인터의 자료형에 따라 참조 단위가 달라짐

실습예제 8-7 ptrtypecast.c

포인터 자료형의 변환

```
01  // file: ptrtypecast.c
02  #include <stdio.h>
03
04  int main(void)
05  {
06     int value = 0x61626364;
07     int *pi = &value;
08     char *pc = (char *) &value;
09
10     printf("변수명    저장값       주소값\n");
11     printf("----------------------------\n");
12     printf(" value   %0#x  %u\n", value, pi); //정수 출력
13
14     //문자 포인터로 문자 출력 모듈
15     for (int i = 0; i <= 3; i++)
16     {
17        char ch = *(pc + i);
18     printf("*(pc+%d)  %0#6x  %2c  %u\n", i, ch, ch, pc + i);
19     }
20
21     return 0;
22  }
```

문자를 6자리의 16진수로 출력

문자를 직접 출력

설명

```
06  int 형 변수 value를 선언하여 16진수 0x61626364을 저장, 0x61은 1바이트인 문자로 해석하면
    문자 'a'이며, 마찬가지로 0x62는 문자 'b'이며, 0x63은 문자 'c'이고, 0x64은 문자 'd'임
07  int 포인터 pi 선언하여 int 변수 value의 주소를 저장
08  char 포인터 pc 선언하여 int 변수 value의 주소를 char 포인터로 형변환하여 저장, 이제 pc는
    char 포인터이므로 1바이트씩 이동 가능
10  출력을 위한 제목 헤드라인 출력
11  출력을 위한 헤드라인 출력
12  변수 value의 변수명 저장값 주소값 출력, 저장값은 16진수로 출력
15  반복 for는 제어변수 i가 0, 1, 2, 3까지 4회 실행
16~19 반복몸체가 두 문장이므로 반드시 블록이 필요
17  char 변수 ch에 (pc+i)가 가리키는 문자 변수를 저장
18  (pc+i)가 가리키는 문자 변수의 변수명 저장값 주소값 출력, 저장값은 16진수와 문자 각각 2번 출력
```

실행결과

```
변수명      저장값        주소값
----------------------------
 value   0x61626364  0045F910
*(pc+0)  0x0064   d  0045F910
*(pc+1)  0x0063   c  0045F911
*(pc+2)  0x0062   b  0045F912
*(pc+3)  0x0061   a  0045F913
```

이중 포인터

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

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

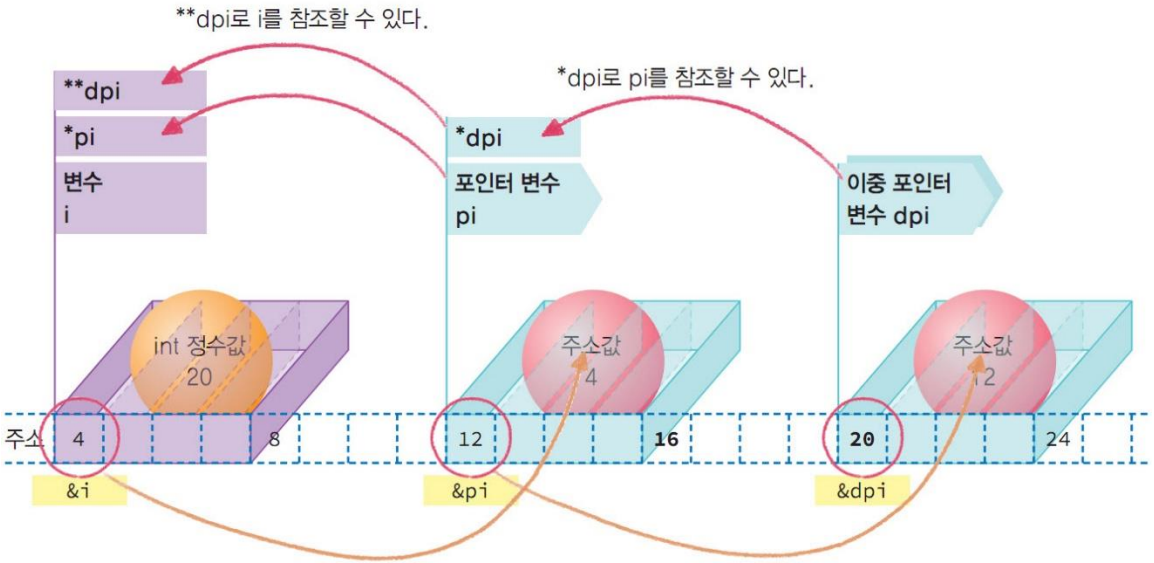


그림 8-15 이중포인터의 메모리와 변수

다중 포인터 예제

예제 multipointer.c

- 포인터와 이중 포인터의 활용

포인터 형변환

- 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

```
*pi = i + 2;          // i = i + 2;
**dpi = *pi + 2;      // i = i + 2;
```

그림 8-16 다중포인터의 이용

실습예제 8-8 multipointer.c

이중 포인터를 이용한 변수의 참조

```
01  // file: multipointer.c
02  #include <stdio.h>
03
04  int main(void)
05  {
06     int i = 20;
07     int *pi = &i;       //포인터 선언
08     int **dpi = &pi;    //이중 포인터 선언
09
10     printf("%p %p %p\n", &i, pi, *dpi);
11
12     *pi = i + 2;    // i = i + 2;
13     printf("%d %d %d\n", i, *pi, **dpi);
14
15     **dpi = *pi + 2;// i = i + 2;
16     printf("%d %d %d\n", i, *pi, **dpi);
17
18     return 0;
19  }
```

모두 변수 i의 주소값을 참조하는 방식이다.

설명

```
06  int 형 변수 i를 선언하여 20 저장
07  int 포인터 pi 선언하여 int 변수 i의 주소를 저장
08  int 이중 포인터 dpi 선언하여 int 포인터 변수 pi의 주소를 저장
10  &i는 i의 주소, pi도 i의 주소, *dpi는 pi이므로 i의 주소이므로 모두 i의 주소가 출력
12  *pi는 변수 i와 같으므로 i = i + 2;와 같음
13  *pi는 변수 i, **dpi도 변수 i이므로 모두 i의 저장값인 22가 출력
15  **dpi는 변수 I, *pi는 변수 i와 같으므로 i = i + 2;와 같음
16  *pi는 변수 i, **dpi도 변수 i이므로 모두 i의 저장값인 24가 출력
```

실행결과

```
003EFD50 003EFD50 003EFD50
22 22 22
24 24 24
```

모두 변수 i의 주소값임

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

표 8-3 연산자 우선순위

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)

  - 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

다양한 포인터 연산

예제 variousop.c

- 간접연산자와 증가연산자의 활용

실습예제 8-9 variousop.c

```
01  // file: variousop.c
02  #include <stdio.h>
03
04  int main(void)
05  {
06     int i;
07     int *pi = &i;        //포인터 선언
08     int **dpi = &pi;     //이중포인터 선언
09
10     *pi = 5;
11     *pi += 1;            //*pi = *pi + 1과 같음
12     printf("%d\n", i);
13
14     // 후위 연산자 pi++는 전위 연산자보다 *pi보다 빠름
15     printf("%d\n", (*pi)++); //*pi++ 는 *(pi++)로 (*pi)++와 다름
16     printf("%d\n", *pi);
17
18     *pi = 10;
19     printf("%d\n", +++pi); //+++pi와 ++(*pi)는 같음
20     printf("%d\n", ++++dpi); //++++dpi와 ++(**dpi)는 같음
21     printf("%d\n", i);
22
23     return 0;
24  }
```

설명

```
06  int 형 변수 i를 선언
07  int 포인터 pi 선언하여 int 변수 i의 주소를 저장
08  int 이중 포인터 dpi 선언하여 int 포인터 변수 pi의 주소를 저장
10  *pi는 변수 i와 같으므로 i = 5와 같음
11  *pi는 변수 i와 같으므로 i = i + 1와 같음
12  변수 I 출력
15  (*pi)++에서 *pi는 변수 i와 같으므로 i++와 같음
16  *pi는 변수 i, 그러므로 i 출력
18  *pi는 변수 i, 그러므로 i = 10과 같음
19  전위연산자의 결합성은 우에서 좌이므로 +++pi는 ++(*pi)로, ++i와 같음
20  전위연산자의 결합성은 우에서 좌이므로 ++++dpi는 ++(**dpi)로, ++i와 같음
21  최종 변수 i 출력
```

실행결과

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

포인터 상수

예제 constptr.c

- 포인터 상수 활용

실습예제 8-10 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 주소값을 저장할 수 없음
   *pd = 4.4;
   printf("%f\n", *pd);

   return 0;
}
```

설명

```
int 형 변수 i와 j를 선언
포인터 변수 p를 선언하면서 간접 연산 *p를 상수로 선언
포인터 변수 p의 내용인 주소값을 j의 주소값으로 수정
*p는 변수 j와 같으므로 20 출력
double 형 변수 d와 e를 선언
포인터 변수 pd를 선언하면서 포인터 pd를 상수로 선언
포인터 변수 pd가 가리키는 변수 d의 내용을 4.4로 수정
*pd는 변수 d와 같으므로 4.4 출력
```

실행결과

```
20
4.400000
```

포인터 형변환

- ❶ *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

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

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

- ❷ 1번과 동일한 문장으로 간접 연산식 *pi를 상수로 만드는 방법
- ❸ 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

LAB 표준입력으로 받은 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

Lab 8-3 sumpointer.c

```
01  // file: sumpointer.c
02  #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
03  #include <stdio.h>
04
05  int main(void)
06  {
07     double x, y;
08     double *px = &x;
09     double *py = &y;
10
11     //포인터 변수 px와 py를 사용
12     printf("두 실수 입력: ");
13     scanf("%lf %lf", _________, ___________);
14     //합 출력
15     printf("%.2f + %.2f = %.2f\n", _________, ___________, ___________);
16
17     return 0;
18  }
```

정답

```
13  scanf("%lf %lf", px, py);
15  printf("%.2f + %.2f = %.2f\n", *px, *py, *px + *py);
```

Thank you