컴퓨터 프로그래밍 I
(CSE2003-3)

Mon/Wed 16:30-17:45 pm

Lecture 20

Pointers

Objectives

- Pointer의 개념과 사용법을 이해해야 한다
- Pointer를 선언(declare)하고 만들고(define), 초기화(initialize)하는 방법을 알아야 한다
- Pointer를 이용하여 데이터에 접근하는 프로그램을 작성할 수 있어야 한다
- Pointer를 parameter로 보내는 방법과 return 값으로 반환하는 방법을 알아야 한다.
- Pointer들 사이의 compatibility 개념을 이해해야 한다

호랑MB . . .

  - 이 chapter가 끝날 때 위의 목표를 달성했는지 반드시 체크해 볼 것!!!

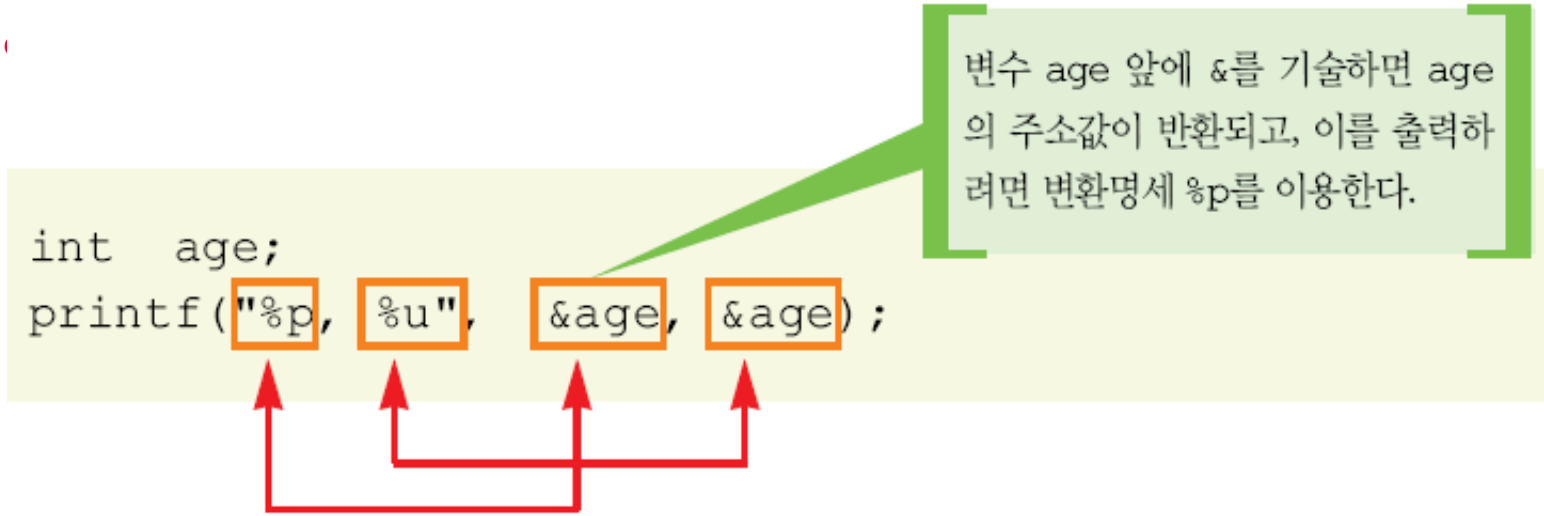
Memory Address

- **Address(주소)**
  - 메모리에는 그 메모리의 저장 장소의 위치를 나타내는 주소 값
  - 주소(address)는 1바이트마다 1씩 증가하도록 메모리에는 연속적인 번호가 구성

그림 14.1 메모리와 주소

주소 연산자

- **&변수**
  - 변수의 주소 값을 알아내려면 변수 앞에 주소 연산자 & (ampersand)를 이용
- **주소 값 이용 장단점**
  - 주소 값을 이용하면 보다 편리하고 융통성 있는 프로그램이 가능



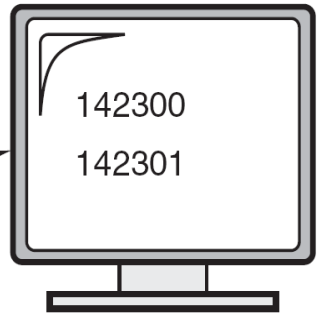
Concepts of pointers

◆ Pointer values

```
// Print character addesses
#include <stdio.h>

int main (void)
{
// Local Declarations
   char a;
   char b;
// Statements
   printf ("%p\n %p\n", &a, &b);
   return 0;
}  // main
```

변수 a, b로 할당된 메모리의 주소

주소 연산자를 이용한 변수 a, b의 출력



- 컴퓨터는 char 변수 a, b가 선언될 때 저장공간을 1byte씩 제공
- 위 프로그램의 결과로 변수 a, b의 주소 값 142300, 142301이 출력

- 자료형 char형 변수와 int형 변수, double형 변수를 각각 선언하여 값을 저장한 후, 저장 값과 주소 값을 각각 출력하는 프로그램
- 변수의 주소 값을 출력하기 위해서는 변환 명세에 %p나 %u 로 출력

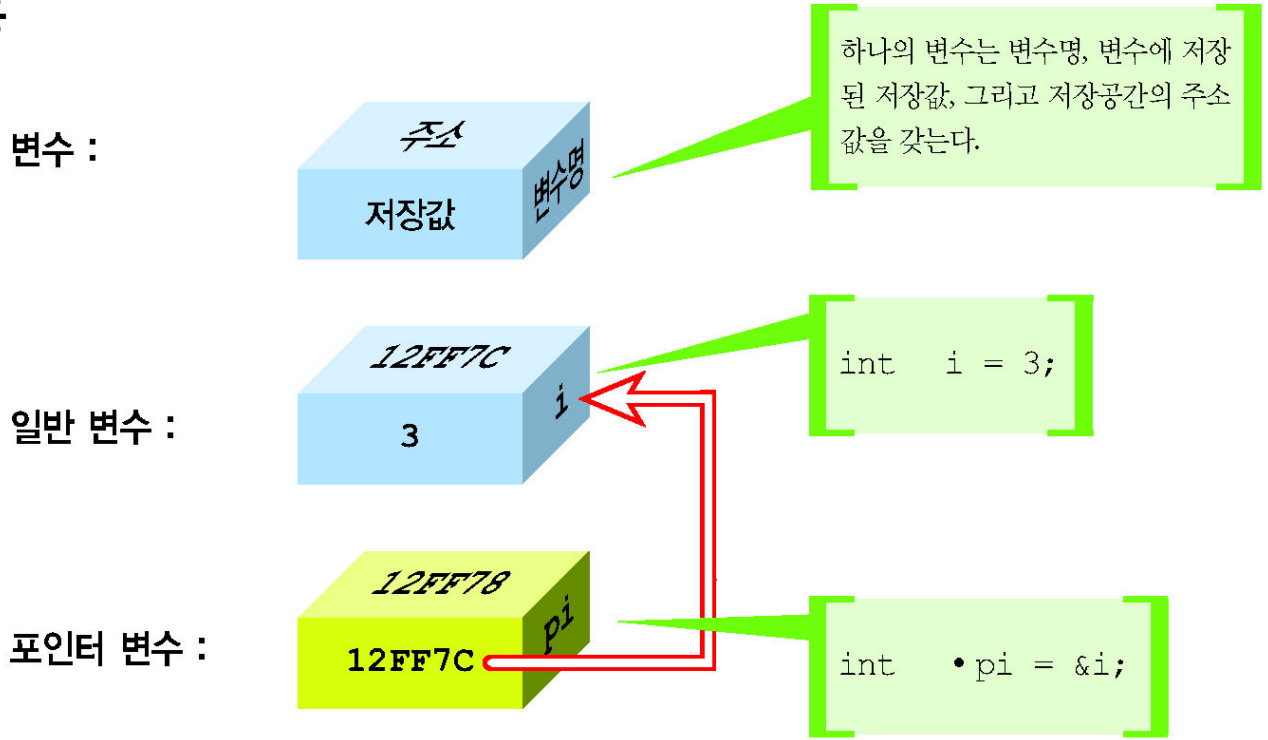
```c
/*******************
  파일 : address.c
 *******************/

#include <stdio.h>

int main(void)
{
    char ch = 'a';
    int  num = 100;
    double real[2] = {3.14, 1.87};

    printf("char   변수(ch)  값 %6c의 주소는 %p이며, 십진수로 %8u이다.₩n", ch, &ch, &ch);
    printf("int    변수(num) 값 %6d의 주소는 %p이며, 십진수로 %8u이다.₩n", num, &num, &num);
    printf("₩n변수 ch와 변수 num의 주소 값의 차이는 %u이다₩n₩n",
           (unsigned)&ch - (unsigned)&num);
    printf("double 변수(real[1]) 값 %6.2f의 주소는 %p이며, 십진수로 %8u이다.₩n",
           real[1], &real[1], &real[1]);
    printf("double 변수(real[0]) 값 %6.2f의 주소는 %p이며, 십진수로 %8u이다.₩n",
           real[0], &real[0], &real[0]);
    printf("₩n변수 numr과 변수 real[0]의 주소 값의 차이는 %u이다₩n₩n",
           (unsigned)&num - (unsigned)&real[0]);

    return 0;
}
/*** End of address.c ***/
```

[출력 창]

```
char   변수(ch)  값      a의 주소는 0012FF7C이며, 십진수로  1245052이다.
int    변수(num) 값    100의 주소는 0012FF78이며, 십진수로  1245048이다.

변수 ch와 변수 num의 주소 값의 차이는 4이다

double 변수(real[1]) 값   1.87의 주소는 0012FF70이며, 십진수로  1245040이다.
double 변수(real[0]) 값   3.14의 주소는 0012FF68이며, 십진수로  1245032이다.

변수 numr과 변수 real[0]의 주소 값의 차이는 16이다

Press any key to continue
```

포인터 변수

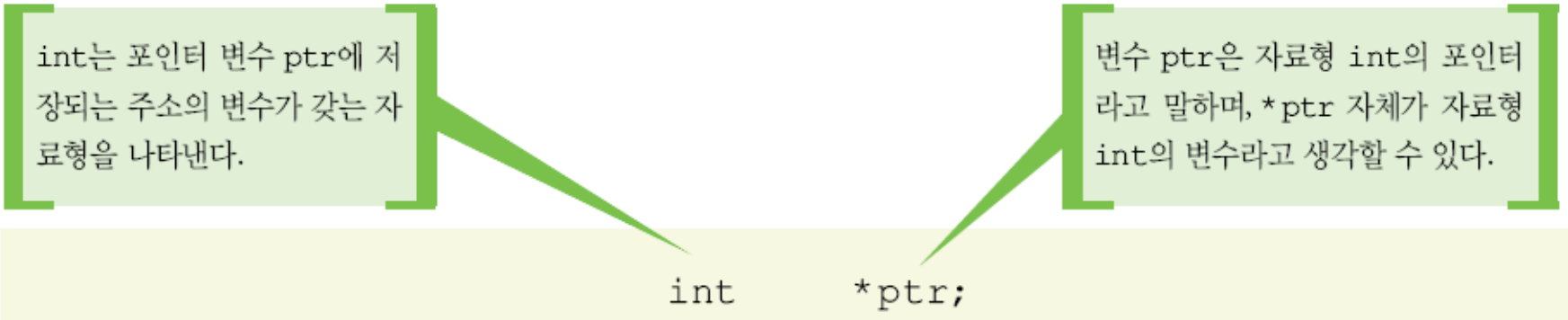
- 포인터 변수는 일반 변수와는 다르게 변수에 저장되는 값이 메모리의 주소(address) 값만을 저장할 수 있는 특별한 변수
- 포인터 변수를 이용하면 프로그램이 간결하고 효율적이므로 포인터 변수를 이용



- 변수 이름은 그 변수가 선언된 함수에서만 볼 수 있지만, 그 변수의 주소를 알면 모든 함수에서 접근 가능
- 포인터 변수는 주소를 값으로 갖는 변수.

포인터 변수 선언

- **포인터 변수의 선언**
  - 포인터 변수는 그 포인터가 가리키는 변수의 자료형에 맞추어 형을 선언해야 함
  - 포인터 변수 임을 나타내 주는 *
  - 별표 *의 위치가 변수 자료형 int와 변수명 사이 어디에 위치하든 관계 없음



- **여러 개의 포인터 변수를 한 번에 선언**
  - 다음은 ptr1만 포인터 변수

```
int*    ptr1, ptr2, ptr3;
```

  - 세 변수를 모두 포인터 변수로 선언하려면 다음과 같이

```
int    *ptr1, *ptr2, *ptr3;
```

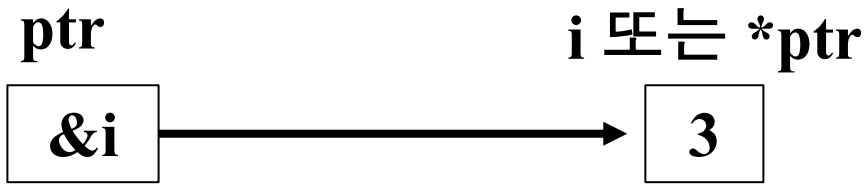
포인터 변수의 값

- **포인터 변수는 주소(address) 값만을 저장**
  - 포인터 변수도 초기 값 지정 가능

```
int  i;            //double phi = 3.14;
int  *ptr = &i;    //double *ptr = &phi;
```

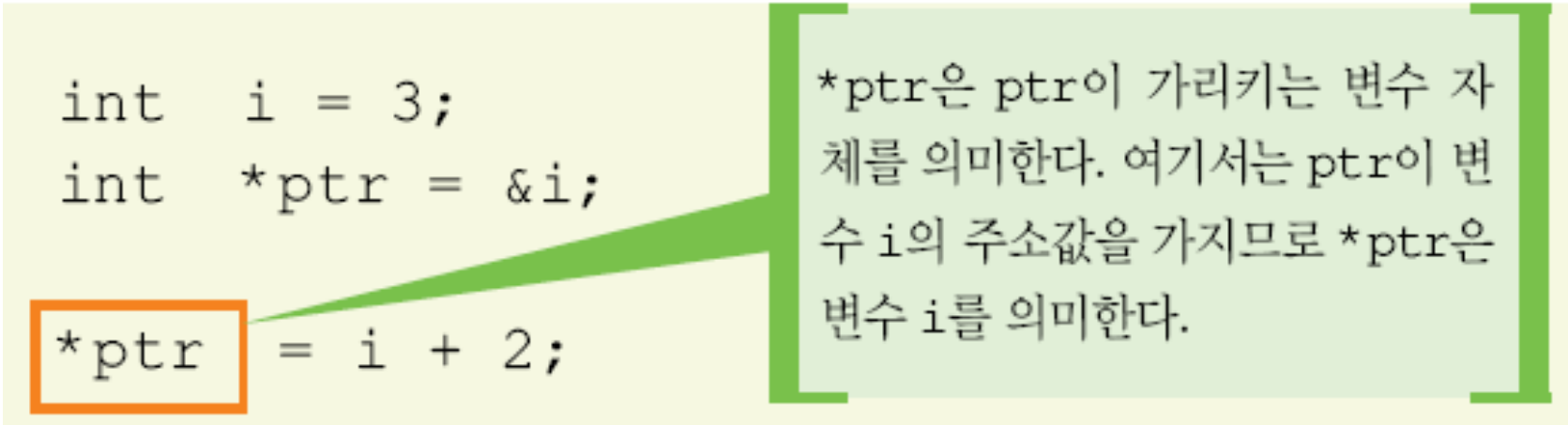
  - 위 구문은 변수 ptr에는 변수 i의 주소 값이 저장
  - 변수 ptr은 초기 값이 변수 i의 주소가 지정되었으므로 *ptr은 변수 I 자체를 의미

```
int  i = 3;
int  *ptr = &i;
```



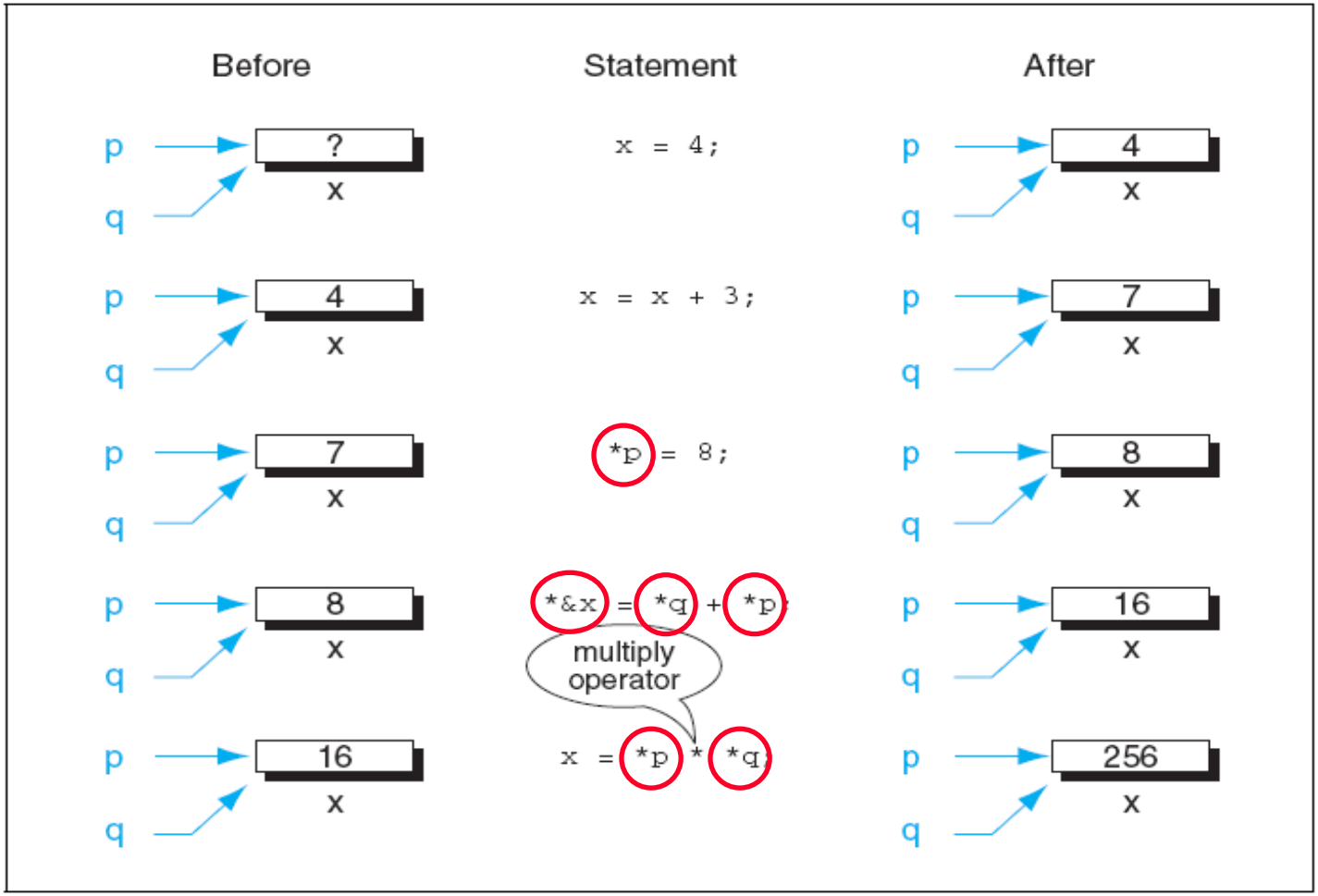
역참조 연산자(Indirection Operator)

- **역참조 연산자(Indirection operator) ***
  - 역참조 연산자는 포인터 변수 앞에 연산자 *를 붙이면 그 포인터가 가리키는 변수를 지칭
- **구문 *ptr = i + 2;을 이용 (i = i + 2; 와 같은 명령)**
  - 변수 i의 값이 2 증가
  - 연산자 *는 포인터 변수를 뒤에 기술하면 역참조(dereference) 연산자



Accessing variables through pointers

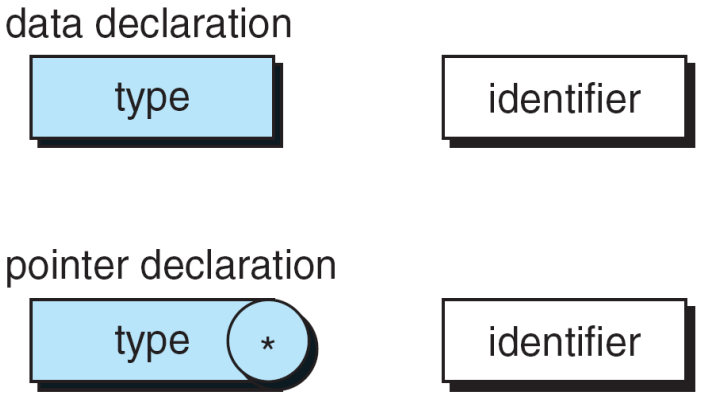
- p, q가 변수 x를 가리키는 포인터 변수일 때, 다음과 같이 변수 x를 access하는 여러 가지 방법이 있다.



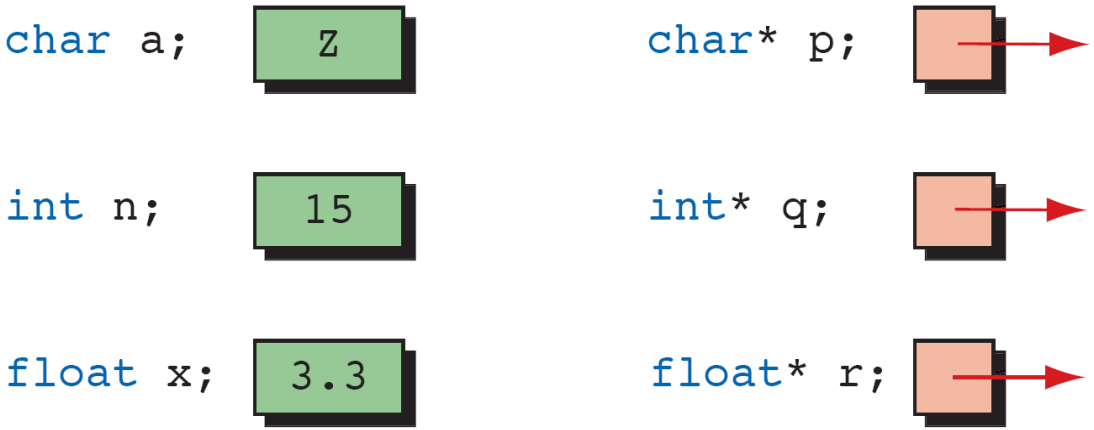
○ 표 한 것은 모두 'x'에 해당한다.

Pointer declaration and definition

- **Pointer variable declaration**



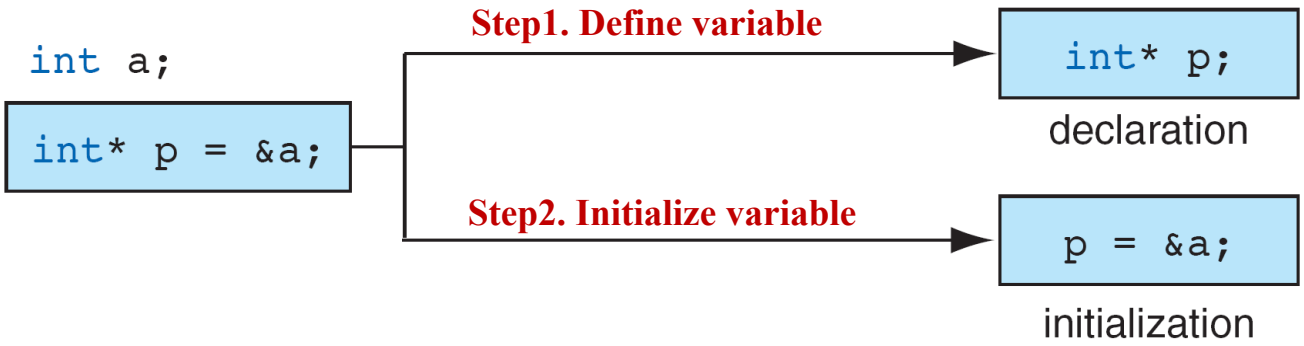
- **서로 다른 타입을 참조하는 포인터 변수의 선언**



Initialization of pointer variables

- 일반적인 변수가 선언된 후에 초기화되지 않으면 쓰레기 값을 갖는 것처럼 포인터 변수도 초기화하지 않으면 엉뚱한 곳을 가리키게 된다.

- 포인터 변수는 주소값을 가지므로 주소 연산자(&)를 이용하여 초기화한다.



- 포인터 변수가 아무 변수도 가리키지 않도록 하려면 다음과 같이 초기화한다.

```
*p = NULL;
```

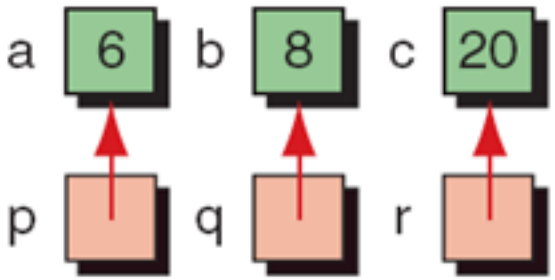
포인터 사용의 예

선언 및 초기화		
int i = 3, j =5, *p = &i, *q = &j, *r; **double x;**		
수 식	**등가 수식**	**값**
p == & i;	**p == (& i);**	**1**
*** * & p;**	*** (* (& p));**	**3**
r = & x;	**r = (& x);**	**/* illegal */**
7 * * p / * q + 7;	**((7 * (* p)) / (* q)) + 7;**	**11**
*** (r = & j) *= *p;**	**(* (r = (& j))) *= (* p);**	**15**

Example program

- 예제 프로그램 – 포인터 변수를 사용한 예제

```c
#include <stdio.h>

int main(void)
{
        int a, b, c;
        int *p, *q, *r;

        a = 6;
        b = 2;
        p = &b;

        q = p;
        r = &c;

        p = &a;
        *q = 8;

        *r = *p

        *r = a + *q + *&c;

        printf("%d %d %d\n", a, b, c);
        printf("%d %d %d\n", *p, *q, *r);

        return 0;
}
```




```
[root@mclab chap9]# vi chap9-2.c
[root@mclab chap9]# gcc -o chap9-2 chap9-2.c
[root@mclab chap9]# ./chap9-2
6 8 20
6 8 20
[root@mclab chap9]#
```

Pointers and functions

- **C는 함수를 호출할 때 기본적으로 "값에 의한 호출(call-by-value)" 메커니즘을 사용한다**
  - caller(호출하는 쪽)에서 callee(호출되는 쪽)에 parameter를 넘길 때 변수의 값을 넘겨주는 방식

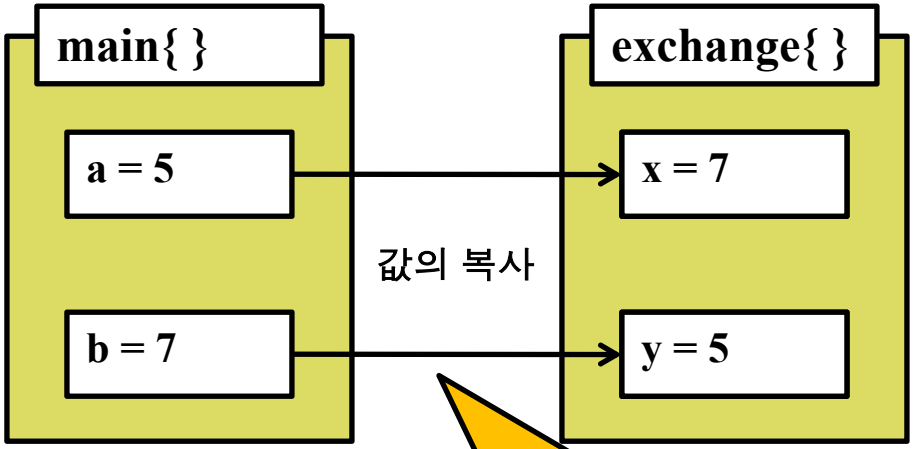
- **하지만 이 방법은 callee가 넘겨받은 parameter의 값을 변경하더라도 caller에 영향을 미치지 못한다.**
  - 그래서 callee에서 caller의 변수를 변경해야 할 필요가 있을 때에는 적절히 동작하지 않는다.

Pointers and functions

- 값에 의한 호출(Call by Value) 로 인해 발생하는 문제

```
/* Prototype Declarations
void exchange (int x, int y);

int main (void)
{
  int  a = 5;
  int  b = 7;
  exchange (a, b);
  printf("%d %d\n", a, b);
  return 0;
} /* main */
```

a, b의 값을 서로 바꿀 것을 기대



exchange( )함수를 호출한 후에도 a와 b의 변수값이 바뀌지 않는다.

```
void exchange (int x,
               int y)
{
  int temp;

  temp = x;
  x    = y;
  y    = temp;
  return;
} /* exchange */
```

x와 y의 값을 서로 바꿈

Pointers and functions

- **참조에 의한 호출(Call-by-reference)**
  - 참조에 의한 호출은 caller에서 callee에 parameter를 넘길 때
    - **변수의 값을 넘겨주는 대신 변수의 주소를 넘겨주는 방식**
  - callee에서 caller의 변수에 대한 주소를 가지고 있기 때문에 callee에서 caller의 변수의 값을 변경할 수 있다.
  - 참조에 의한 호출을 사용하면 앞에서와 같은 문제를 피할 수 있다.

- **"참조에 의한 호출"의 효과를 얻는 방법**
  - 함수(callee)의 parameter를 포인터형으로 선언
  - Caller에서 함수(callee)를 호출할 때 parameter로 주소를 전달
  - 함수(callee) 내부에서 parameter 사용시 역참조 연산자(*) 사용
    - 넘겨받은 값이 주소값이므로 그 값을 사용할 때는 당연히 역참조 연산자를 사용해야 한다.

Pointers and functions

- Call by reference 방법을 이용하여 해결한 방법

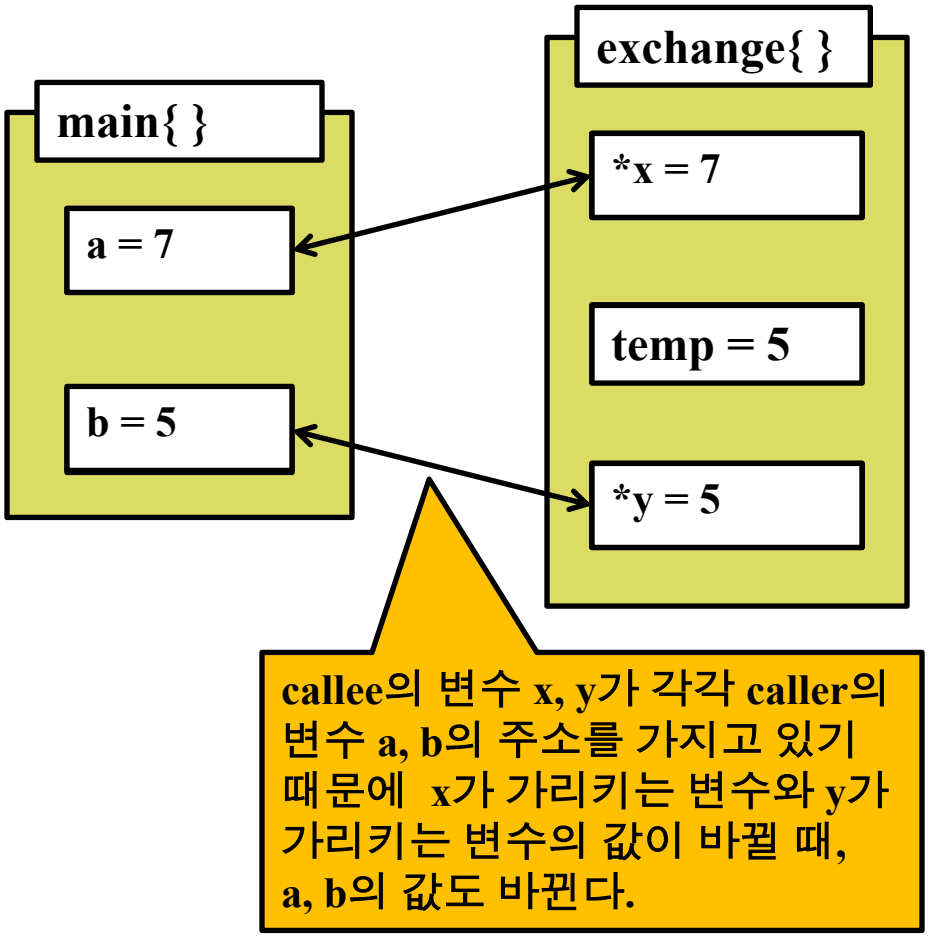
```
/* Prototype Declarations
void exchange (int *, int *);

int main (void)
{
  int  a = 5;
  int  b = 7;
  exchange (&a, &b);
  printf("%d %d\n", a, b);
  return 0;
} /*
```

a, b의 주소를 넘겨준다

포인터 변수를 parameter로 사용

```
void exchange (int *x,
               int *y)
{
  int temp;

  temp = *x;
  *x   = *y;
  *y   = temp;
  return;
} /* excha
```

역참조 연산자(*)를 사용



Pointers and functions

- 예제 프로그램 - Call by reference

```c
#include <stdio.h>

void exchange(int *, int *);

int main(void)
{
        int a = 5;
        int b = 7;
        printf("<<Before>>\n");
        printf("Value   : a=%d b=%d\n", a, b);
        printf("Address : a=%d b=%d\n\n", &a, &b);
        exchange(&a, &b);
        printf("<<After>>\n");
        printf("Value   : a=%d b=%d\n", a, b);
        return 0;
}

void exchange(int *x, int *y)
{
        int temp;

        printf("<<In Function>>\n");
        printf("Value   : x=%d y=%d temp=%d\n", *x, *y);
        printf("Address : x=%d y=%d\n\n", x, y);
        temp = *x;
        printf("Step1(temp=*x) : *x=%d *y=%d temp=%d\n", *x, *y, temp);
        *x = *y;
        printf("Step2(*x=*y)   : *x=%d *y=%d temp=%d\n", *x, *y, temp);
        *y = temp;
        printf("Step3(*y=temp) : *x=%d *y=%d temp=%d\n\n", *x, *y, temp);

        return;
}
```

```
<<Before>>
Value   : a=5 b=7
Address : a=-1076611136 b=-1076611140

<<In Function>>
Value   : x=5 y=7 temp=-1076611176
Address : x=-1076611136 y=-1076611140

Step1(temp=*x) : *x=5 *y=7 temp=5
Step2(*x=*y)   : *x=7 *y=7 temp=5
Step3(*y=temp) : *x=7 *y=5 temp=5

<<After>>
Value   : a=7 b=5
```

주소를 그대로 넘겨받는다.

함수 수행 후 a, b 의 값이 바뀐다.

Pointers and functions

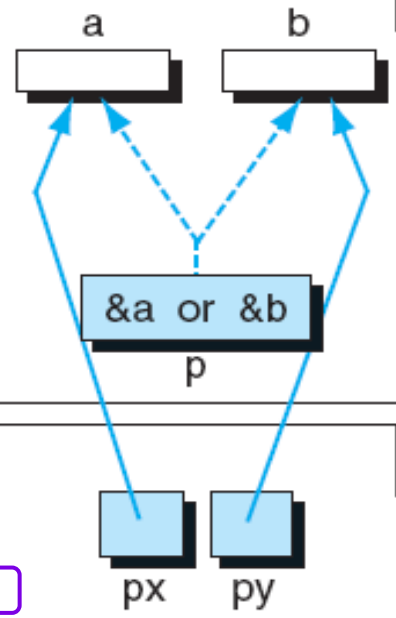
- **Functions returning pointer**
  - 다음 프로그램과 같이 함수가 pointer 변수를 리턴할 수도 있다.
  - 이 경우에는 함수의 헤더 부분에서 return type을 pointer형으로 해주어야 한다.
  - 지역 변수에 대한 참조를 리턴하는 것은 심각한 오류를 불러올 수 있다.

Return type이 pointer형이다.

Pointer type을 return한다. (조건식의 결과에 따라 작은 값을 가진 변수에 대한 참조를 return한다.)

```
/* Prototype Declarations */
int *smaller (int *p1, int *p2);

int main (void)
  …
  int    a;
  int    b;
  int   *p;
  …
  scanf ( "%d %d", &a, &b );
  p = smaller (&a, &b);
  …
```

```
int *smaller (int *px,
              int *py)
{
  return (*px < *py ? px : py);
} /* smaller */
```



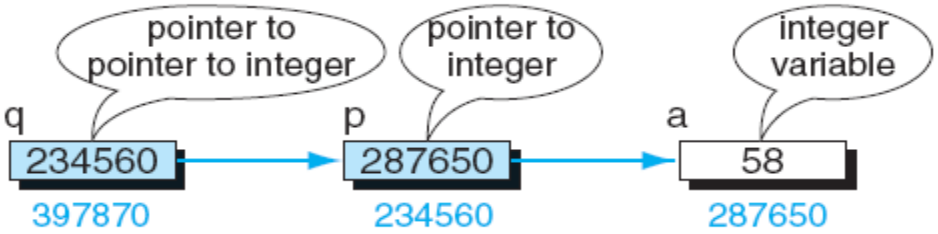
Pointers to pointer

- **포인터 변수가 다른 포인터 변수를 참조할 수도 있다.**
  - 다음 프로그램은 int형 변수 a를 참조하는 포인터 변수 p와, 포인터변수 p를 참조하는 포인터변수 q에 대한 예제이다.

Pointer변수를 참조하는 pointer 변수의 선언

p는 a를, q는 p를 참조한다.

```
/* Local Declarations */
int      a;
int     *p;
int    **q;
/* Statements */
a = 58;
p = &a;
q = &p;
printf(" %3d",   a);
printf(" %3d",  *p);
printf(" %3d", **q);
```

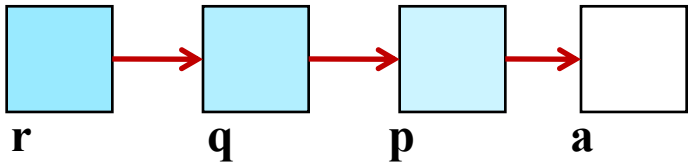


- **integer형 포인터 변수를 참조하는 포인터 변수 q의 선언 방법은 다음과 같다.**
  - **int **q;**
- **q는 p를 통해 a를 참조할 수 있다.**
- **q를 이용하여 a를 참조하기 위해서는 두 단계를 거쳐야 하므로 **q와 같이 역참조 연산자를 두 번 사용한다.**
- **따라서 a, *p, **q에 의해 출력되는 결과는 모두 58 이다.**

Pointers to pointer

- 예제 프로그램 - 포인터의 포인터를 사용한 예

```c
#include <stdio.h>

int main(void)
{
        int a;
        int *p;
        int **q;
        int ***r;

        p = &a;
        q = &p;
        r = &q;

        printf("Enter a number : ");
        scanf("%d", &a);
        printf("\nThe number is %d\n", a);

        printf("\nEnter a number : ");
        scanf("%d", p);
        printf("\nThe number is %d\n", a);

        printf("\nEnter a number : ");
        scanf("%d", *q);
        printf("\nThe number is %d\n", a);

        printf("\nEnter a number is : ");
        scanf("%d\n", **r);
        printf("\nThe number is %d\n", a);

        return 0;
}
```



scanf의 입력 값은 모두 a에 저장된다.

```
[root@mclab chap9]# vi chap9-3.c
[root@mclab chap9]# gcc -o chap9-3 chap9-3.c
[root@mclab chap9]# ./chap9-3
Enter a number : 1
The number is 1

Enter a number : 2
The number is 2

Enter a number : 3
The number is 3

Enter a number : 4
The number is 4
[root@mclab chap9]#
```

Compatibility

- **포인터 변수는 선언될 때 어떤 type의 변수를 참조할 지가 미리 정해진다. 즉, 포인터 변수도 type을 갖는다.**
  - 예제 프로그램 - char, int, double type에 대한 포인터 변수의 size를 출력

```c
#include <stdio.h>

int main(void)
{
        char c;
        char *pc;
        int sizeofc      = sizeof(c);
        int sizeofpc     = sizeof(pc);
        int sizeofStarpc = sizeof(*pc);

        int a;
        int *pa;
        int sizeofa      = sizeof(a);
        int sizeofpa     = sizeof(pa);
        int sizeofStatpa = sizeof(*pa);

        double x;
        double *px;
        int sizeofx      = sizeof(x);
        int sizeofpx     = sizeof(px);
        int sizeofStarpx = sizeof(*px);

        printf("sizeof(c)   : %3d | ", sizeofc);
        printf("sizeof(pc)  : %3d | ", sizeofpc);
        printf("sizeof(*pc): %3d\n", sizeofStarpc);

        printf("sizeof(a)   : %3d | ", sizeofa);
        printf("sizeof(pa)  : %3d | ", sizeofpa);
        printf("sizeof(*pa): %3d\n", sizeofStarpa);

        printf("sizeof(x)   : %3d | ", sizeofx);
        printf("sizeof(px)  : %3d | ", sizeofpx);
        printf("sizeof(*px): %3d\n", sizeofStarpx);

        return 0;
}
```

```
[root@mclab chap9]# vi chap9-4.c
[root@mclab chap9]# gcc -o chap9-4 chap9-4.c
[root@mclab chap9]# ./chap9-4
sizeof(c)   :   1 | sizeof(pc)  :   4 | sizeof(*pc):   1
sizeof(a)   :   4 | sizeof(pa)  :   4 | sizeof(*pa):   4
sizeof(x)   :   8 | sizeof(px)  :   4 | sizeof(*px):   8
[root@mclab chap9]#
```

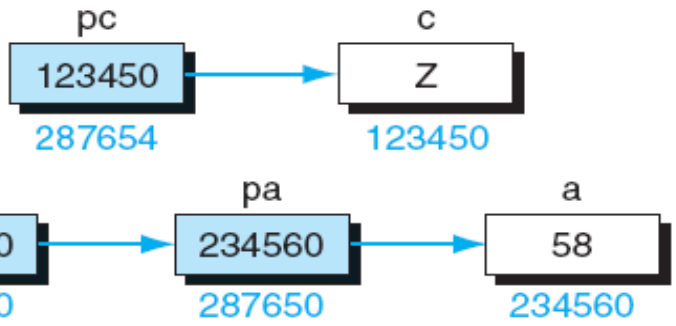
Compatibility

- **Pointer 변수는 참조하는 type에 상관 없이 모두 4byte이다.**
  - 참조하는 변수에 상관 없이 주소를 저장할 크기를 갖는다.

- **앞의 프로그램은 각각의 포인터를 초기화하지 않았지만...**
  - 각각의 포인터가 참조하는 곳의 size는 서로 다르게 지정되어 있음 (1, 4, 8).
  - 이유는 각각의 포인터가 선언될 때, 참조할 변수의 type이 정해져 있기 때문이다.

Compatibility

- 포인터 변수에 다른 타입의 주소를 저장하면 error가 발생한다.



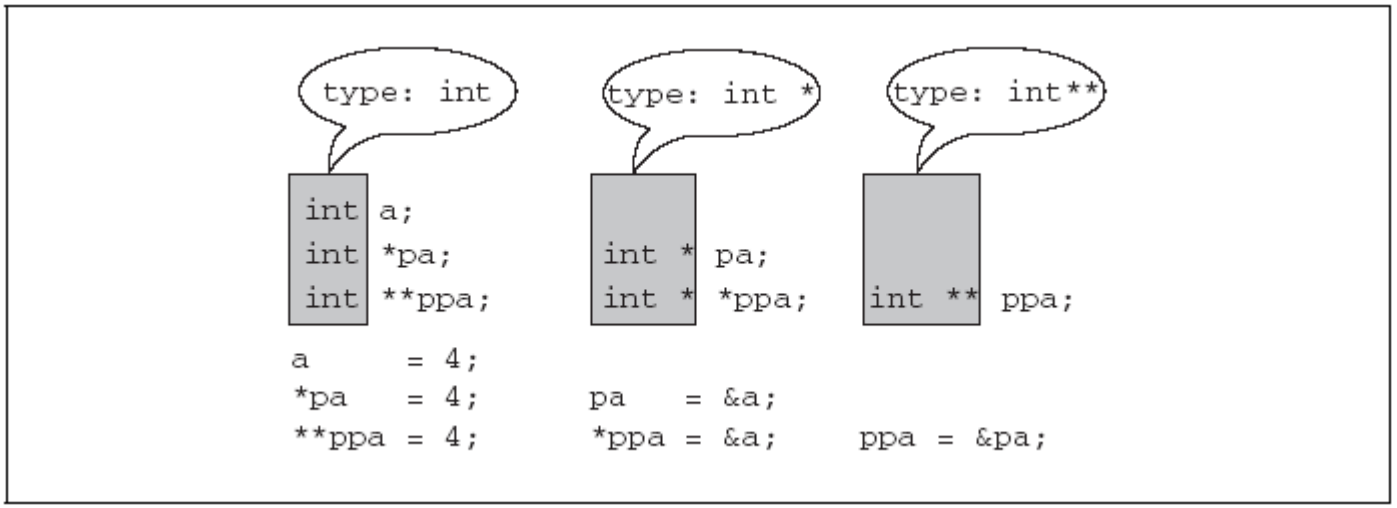

```
char c;
char *pc;

int   a;
int  *pa;
int **ppa;

pc  = &c;                  /* Good and valid    */
pa  = &a;                  /* Good and valid    */
ppa = &pa;                 /* Good and valid    */

/* Invalid pointers will generate errors     */
pc  = &a;                  /* Different types   */
ppa = &a;                  /* Different levels  */
```

Char 타입 포인터에 int형 변수의 주소를 저장 ⟶ error!

int * 타입 포인터에 int형 변수의 주소를 저장 ⟶ error!

Compatibility

- **Pointer types must match !!**



Compatibility

- **Void pointer**
  - 포인터 변수도 type을 가지고 있기 때문에 다른 type의 변수를 참조하도록 한다면 compile error가 발생한다.
  - Void pointer는 임의의 type을 갖지 않기 때문에 어떤 type이든 참조할 수 있다.
  - 선언 방법 : void *pVoid;

- **Casting pointer**
  - 선언된 포인터 변수가 다른 타입의 변수를 참조할 수 있도록 강제적인 형 변환이 가능하다.
  - Ex)
```
int a;
char *p;
p = (char *)&a;
```
  - 이러한 형 변환은 메모리의 낭비를 불러올 수 있다.

Compatibility

- **Casting pointer**
  - 다음은 void pointer를 이용한 변수의 참조와 casting을 통한 참조의 예제이다.

```
/* Local Definitions */
   void *pVoid;
   char *pChar;
   int  *pInt;
/* Statements */
   pVoid  = pChar;
   pInt   = pVoid;
   pInt   = (int *) pChar;
```

  - 둘 다 pChar을 pInt에 저장하고 있다.
    - 첫 번째 경우는 다른 type의 pointer 변수라도 참조가 가능한 void 타입 포인터를 이용했다.
    - 두 번째 경우는 type casting을 이용했다.

Example program

- 다음은 초를 시간으로 바꾸는 함수이다.

```
int secToHours(long time, int *hours, int *minutes, int *seconds)
{
        long localTime;

        localTime = time;
        *seconds  = localTime % 60;
        localTime = localTime / 60;

        *minutes  = localTime % 60;
        *hours    = localTime / 60;

        if(*hours > 24)
                return 0;
        else
                return 1;
}
```

시간, 분, 초 3개의 값을 caller에게 알려줘야 하는데, return문은 하나의 값만을 반환할 수 있으므로 call-by-reference를 이용한다.

Parameter인 역참조 연산자를 이용하여 hours, minutes, seconds가 참조하는 곳에 저장하면 caller쪽의 변수에 저장 된다.