9. Functions

■ Keyword: return

■ pointer/address operators: \* (unary), & (unary)

■ Functions을 정의하는 방법

■ functions의 arguments와 return values를 사용하는 방법

■ function arguments로서 pointer variables를 사용하는 방법

■ Function types

■ ANSI C prototypes

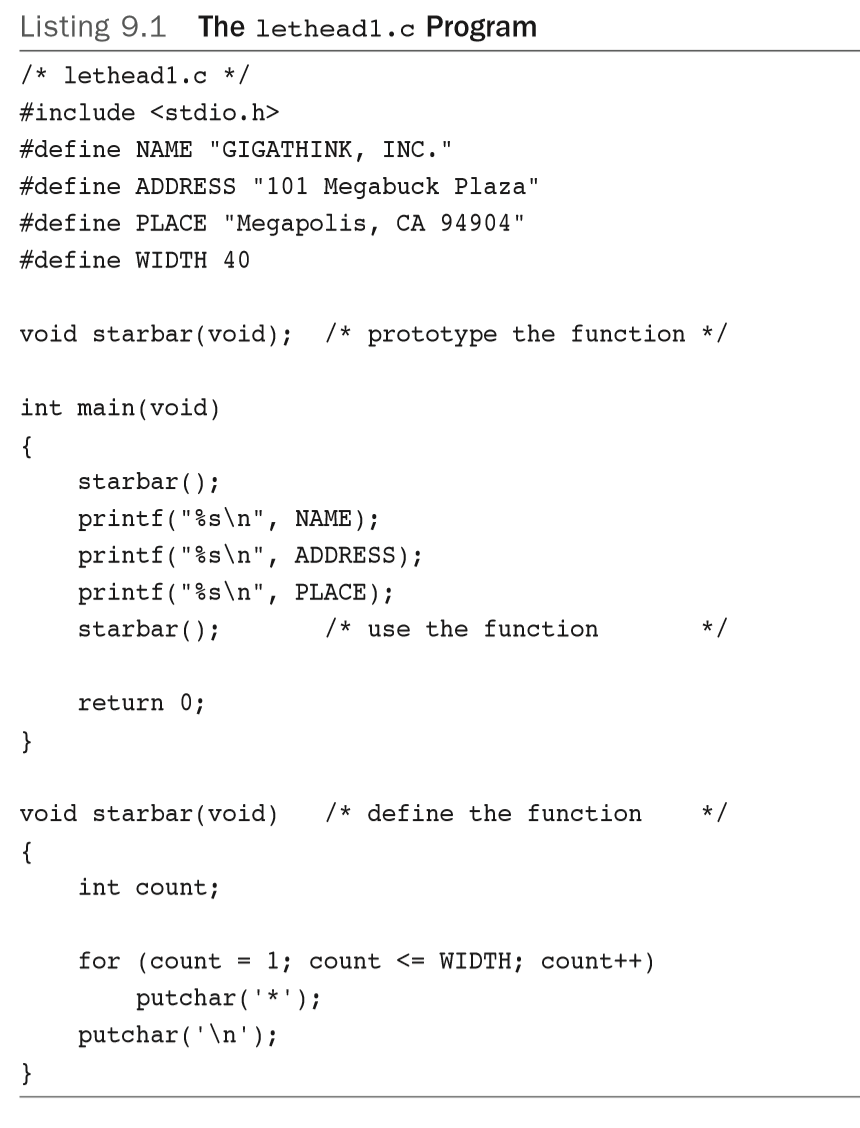
■ Recursion

- functions을 프로그램의 building blocks으로 사용

9.1 Reviewing Functions

* function은 특정 문제를 해결하기 위한 프로그램 code의 독립적인 단위(self-contained unit)
* 프로그램에서 특정 task를 반복하여 실행하려면, 해당 function을 만들어 사용.
* function을 “black box”로 생각
* functions에 대하여 알아야 할 것?
  + function을 어떻게 잘 define하는 방법,
  + function을 이용하기 위하여 call하는 방법
  + functions간에 communication을 하는 방법

9.1.1 a Simple Function을 만들고 사용하기

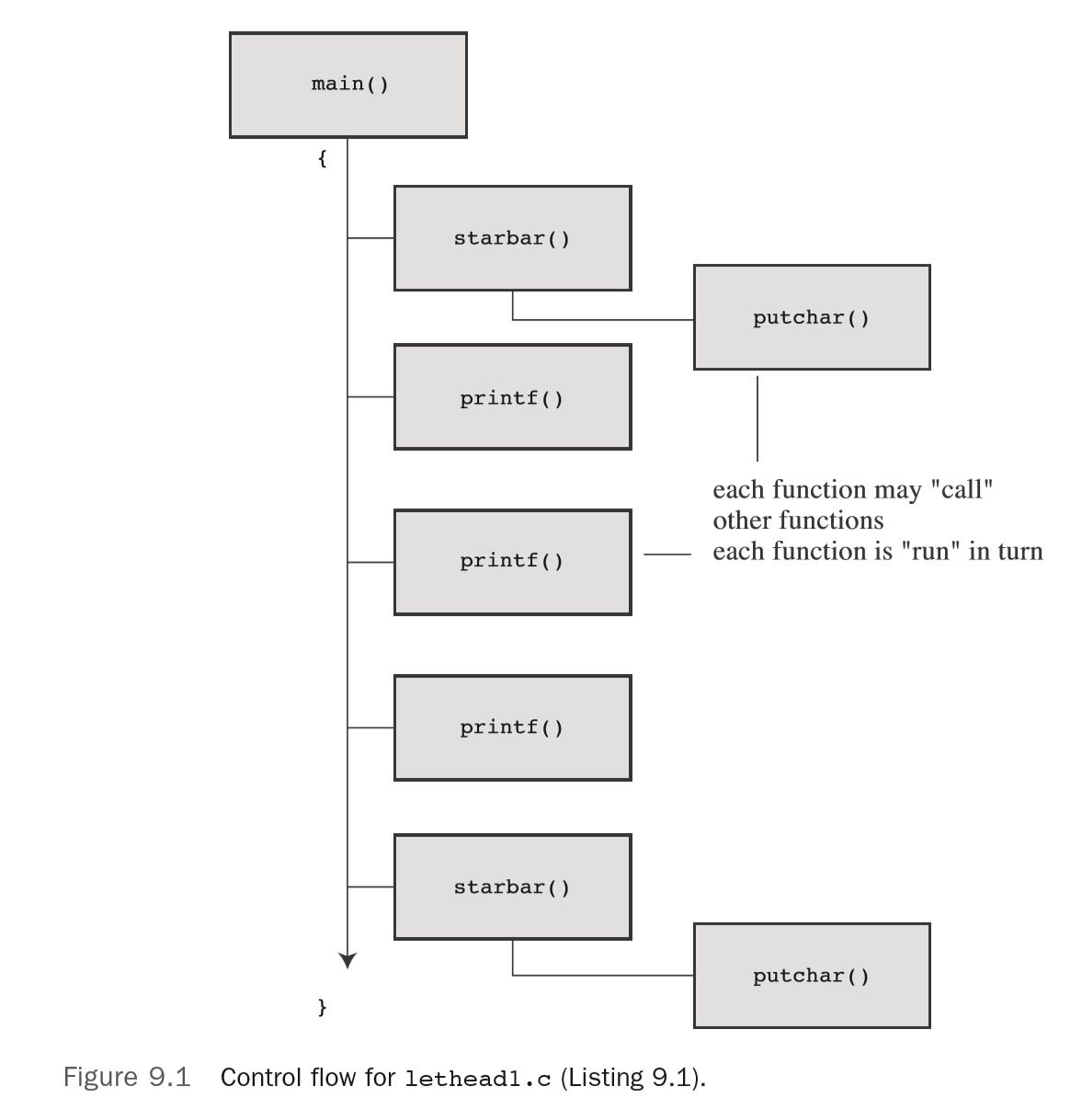


9.1.2 function에 관련된 용어

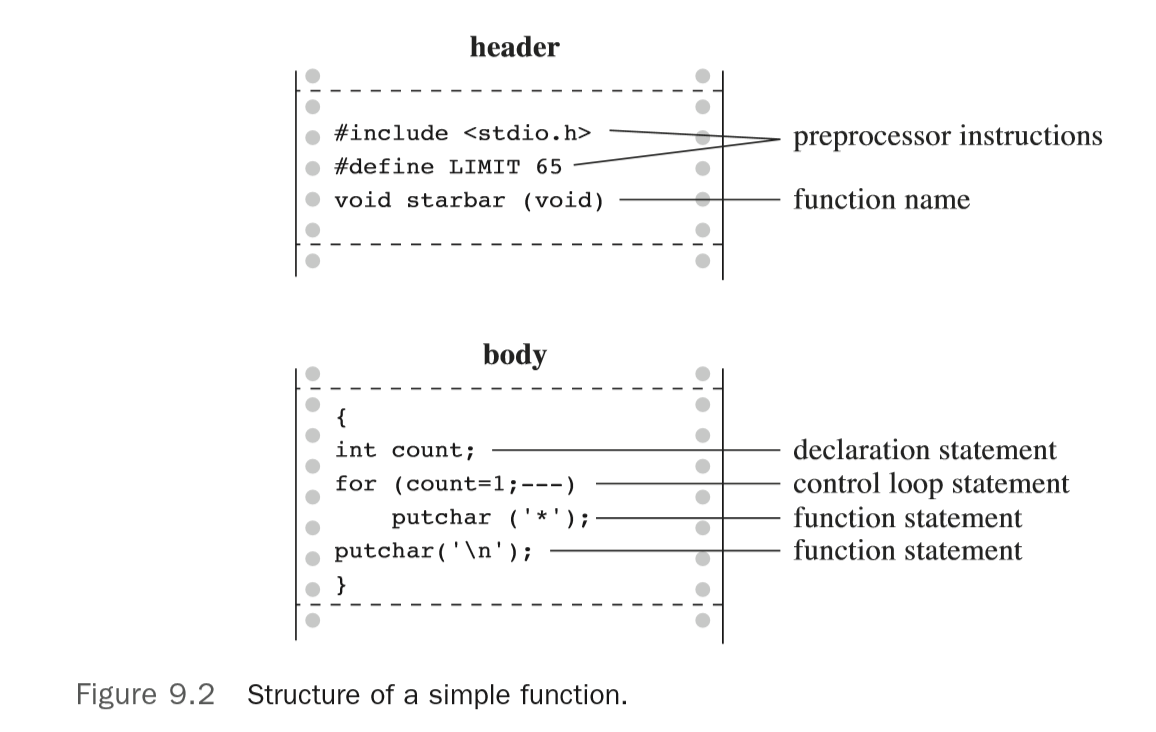
* function prototype : compiler에게 어떤 종류의 function인지를 알려주는 것, void starbar(void);
* function call : function이 실행되도록 호출하는 것,
* function definition : function이 무엇을 하는 것인지를 code를 specify하는 것
* functions는 types을 갖는다
  + function은 사용되기 전에 function에 대한 type을 선언해야 한다.

void starbar(void);

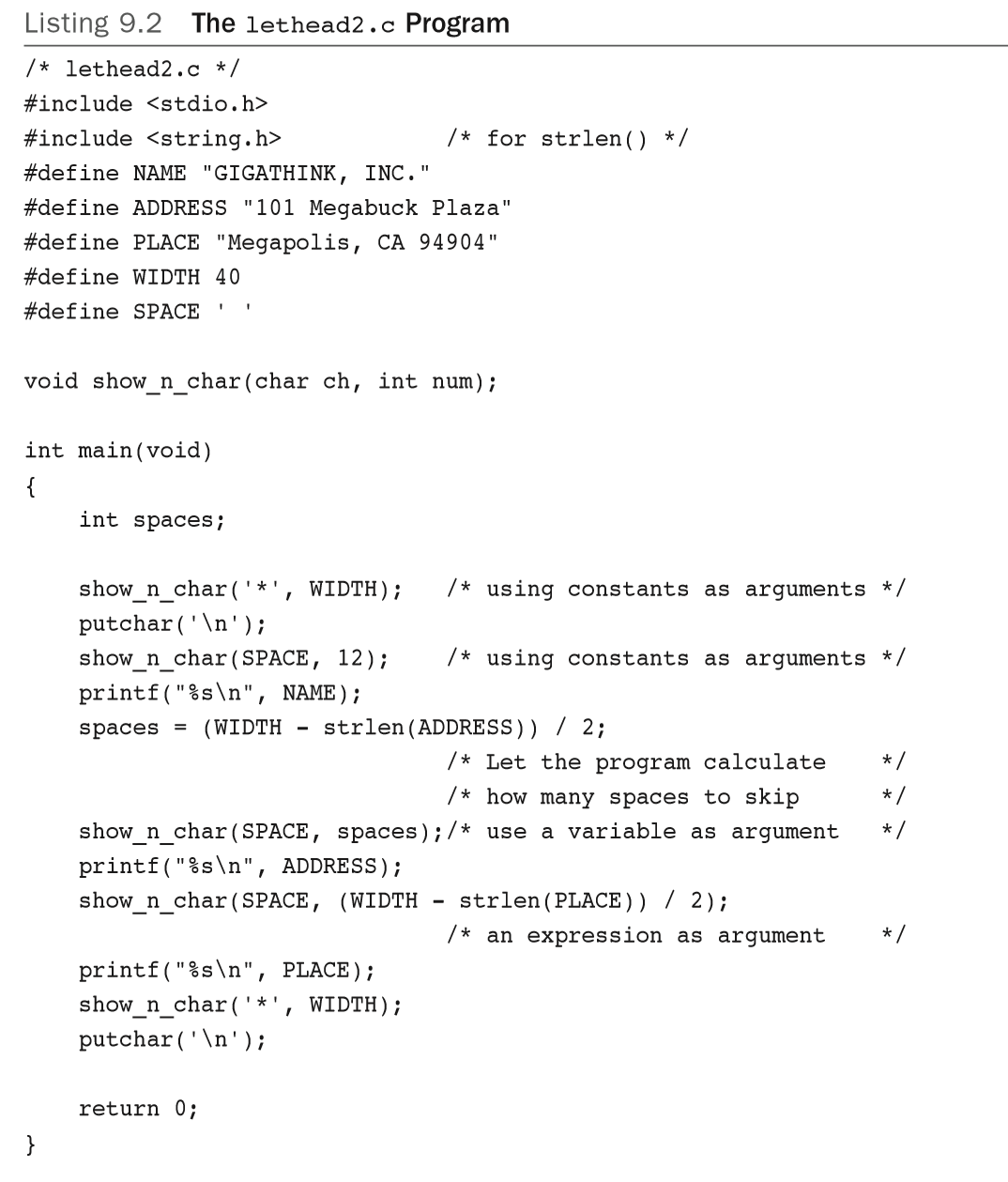
* + 괄호는 starbar가 function name이라는 것을 나타내는 것
  + 첫째 void 는 function type을 나타냄
  + 둘째 void (괄호안에 있는 type)는 function이 no arguments라는 것을 나타냄.
  + semicolon은 function을 선언한다는 것을 나타냄, function을 define하는 것이 아님
* function prototype은 function return 값의 type과 전달되는 arguments의 type을 명시한다.
  + function signature : function 이름, function type, function argument type을 명시하는 것

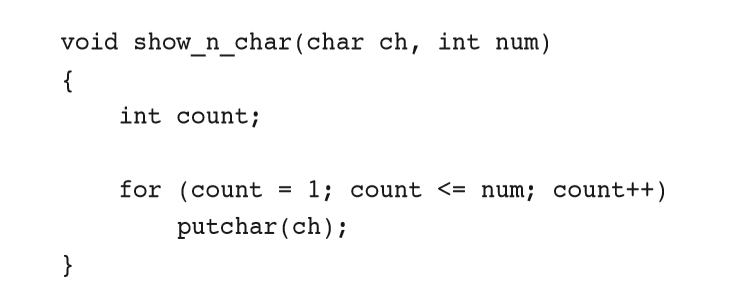


* function starbar()내부에서 선언된 변수 count는 local variable.
  + count의 값은 오직 starbar() 내부에서만 적용
  + main()에서 선언된 count 변수는 전혀 관계가 없음



9.1.3 Function Arguments





9.1.4 Function Argument의 정의: Formal Parameters

void show\_n\_char(char ch, int num)

* show\_n\_char()은 두개의 arguments을 사용: ch, num ,
  + ch: type char , num : type int
  + ch와 num variables을 formal arguments 또는 (다음 용어를 더 선호하여 사용) formal parameters라 부름
  + function내에 정의된 local variable 처럼, formal parameters도 local variables이라 함(private to the function: function 내부의 개인 정보에 해당)
* 표준 ANSI C는 function parameters는 type+변수명을 함께 명시할 것을 요구

void dibs(int x, y, z) /\* invalid function header \*/

void dubs(int x, int y, int z) /\* valid function header \*/

9.1.5 Argument로 Function prototype

- function 사용전에 function prototype을 선언하는 것을 습관화하고 ANSI C prototype 을 준수

void show\_n\_char(char ch, int num); //ANSI C prototype 표준 코딩

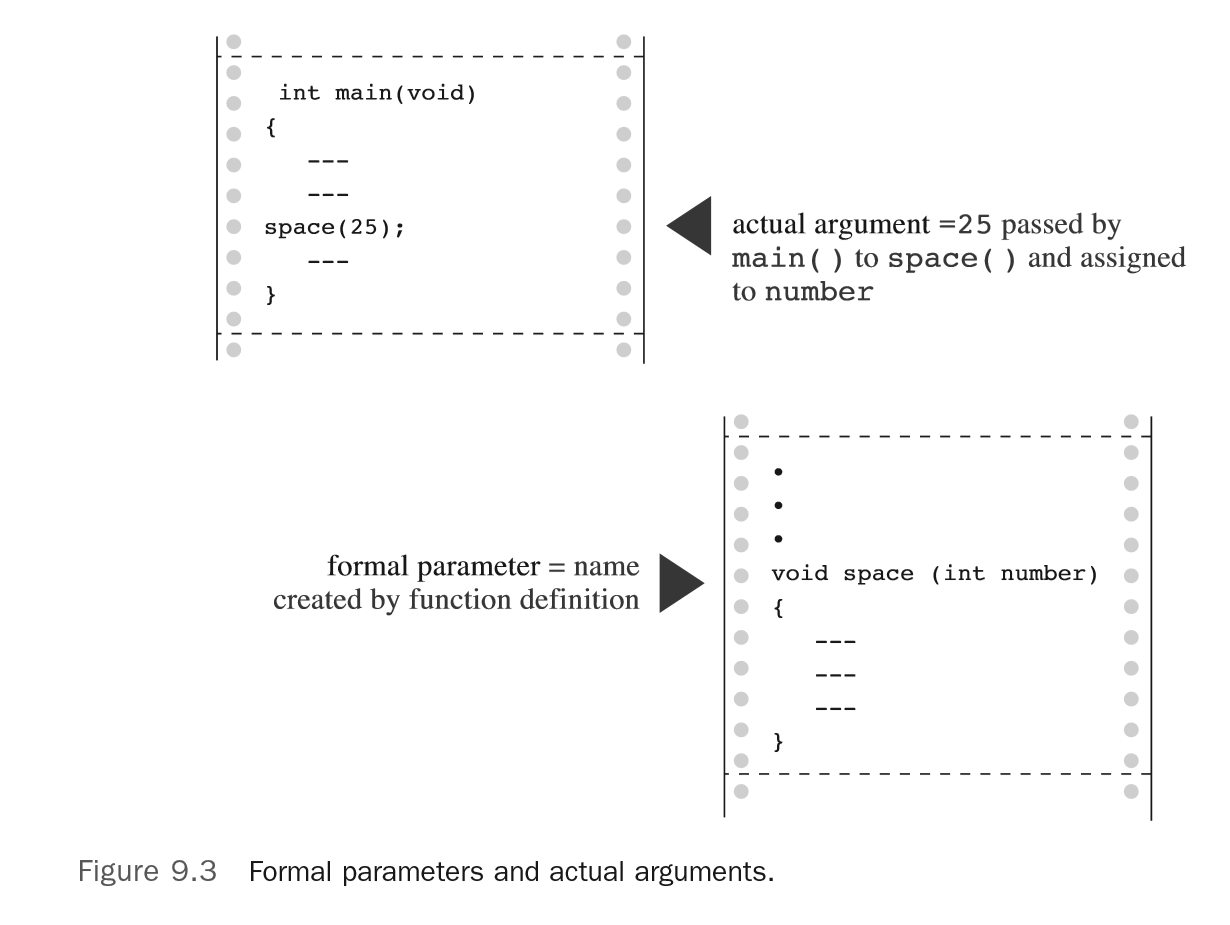
void show\_n\_char(char, int);

9.1.6 Argument를 포함한 Function 호출: Actual Arguments

- function call에서 ch와 num는 actual arguments.

show\_n\_char(SPACE, 12);

* + formal parameter는 호출된 function의 변수,
  + actual argument는 호출 함수에 전달되는 변수 값



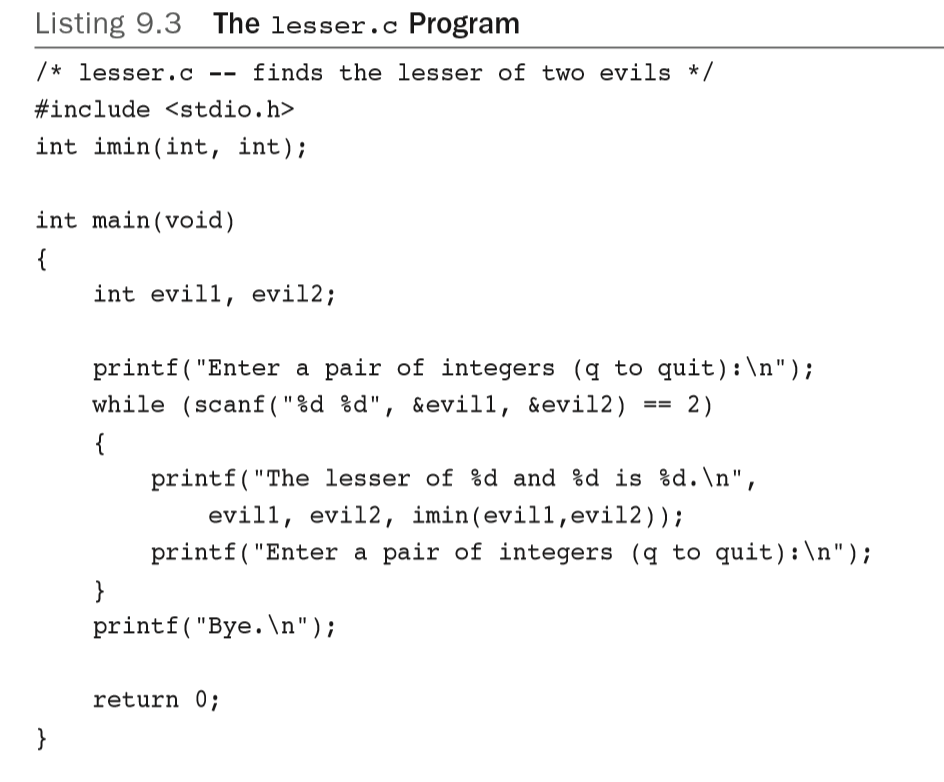
9.1.6 Black-Box 관점

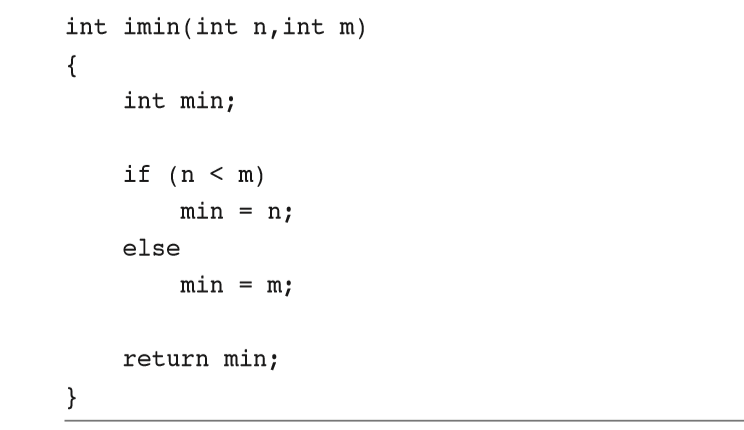
* function을 black box라고 보는 것: function 호출시에 function body가 숨겨졌다는 의미.
* **ch, num, count는 local variables(private to the show\_n\_char() function)**
* **호출된 function의 local variable, formal parameter는 숨겨짐:** black box approach

9.1.7 return을 사용하여 function에서 값을 Returning

- calling function에서 called function으로 값이 전달되는 communicate하는 방법을 알아야 한다.

- functions을 호출하여 test하는 프로그램: driver이라 한다





* keyword return은 호출된 function의 return value을 명시하는 code
* local variable min은 imin()에 private하다고 함 , min 값을 main()에 주려면 return을 사용하여 calling function에 communicated back해야 한다
  + 기억할 것: imin()의 변수들은 local to imin() .
  + imin()의 반환 값을 변수 lesser에 치환:

lesser = imin(n,m);

* + 다음 code는 성립하지 않는다

imin(n,m);

lesser = min;

* function 값은 expression의 일부로서 사용 가능.

answer = 2 \* imin(z, zstar) + 25;

printf("%d\n", imin(-32 + answer, LIMIT));

* return value 값은 variable이 아닌 expression 표현도 가능

/\* minimum value function, second version \*/

imin(int n, int m) {

return (n < m) ? n : m;=>퀴즈

}

* function이 function type과 다른 변수 type으로 반환하는 경우

int what\_if(int n) { //function type은 int

double z = 100.0 / (double) n;

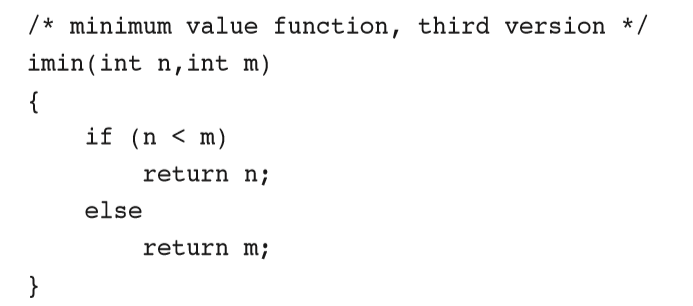
return z; // return 값은 double

}

* 다음 function call 경우:

result = what\_if(64);

호출 function내에서는 1.5625로 치환 . return 문장은 int 값 1를 반환.



9.1.8 Function Types

* Functions은 type으로 선언되어야 한다: function type

#include <stdio.h>

int imin(int, int); /\* imin() declaration \*/

int main(void) {

int evil1, evil2, lesser;

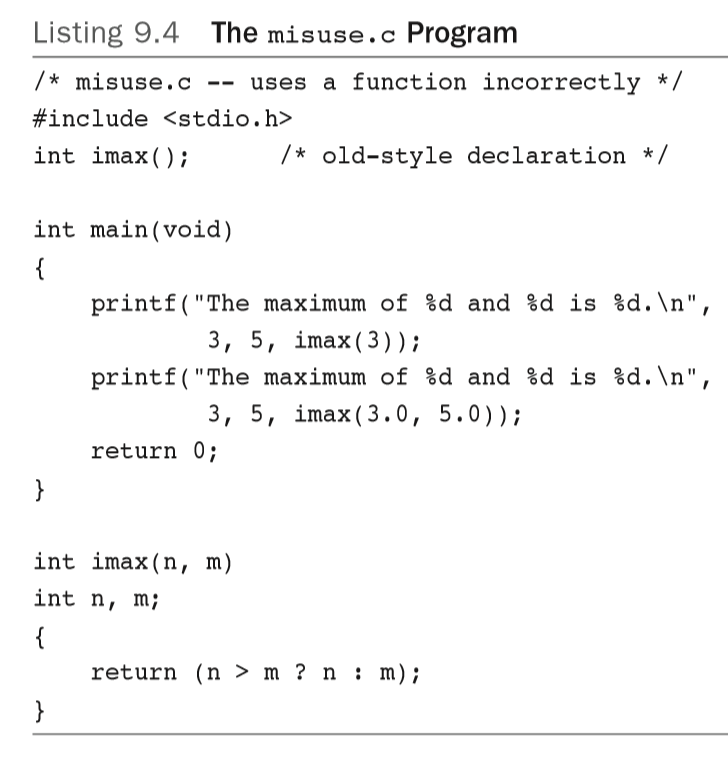
9.2 ANSI C Function Prototyping

* function’s return type을 명시해야 한다, 그러나 function arguments는 생략 가능.

int imin();

* function imin() 호출시에 arguments의 갯수와 type이 일치않아도 compiler는 오류를 내지 않음

9.2.1 Problem



* calling function은 function arguments를 임시 storage area인 stack에 저장
  + called function는 stack으로 부터 arguments를 꺼내온다.
* calling function는 어떤 type이 actual arguments에 전달될지를 결정
* called function은 formal arguments의 type에 기초하여 read(stack에서 가져온다).

9.2.2 ANSI C의 해결책

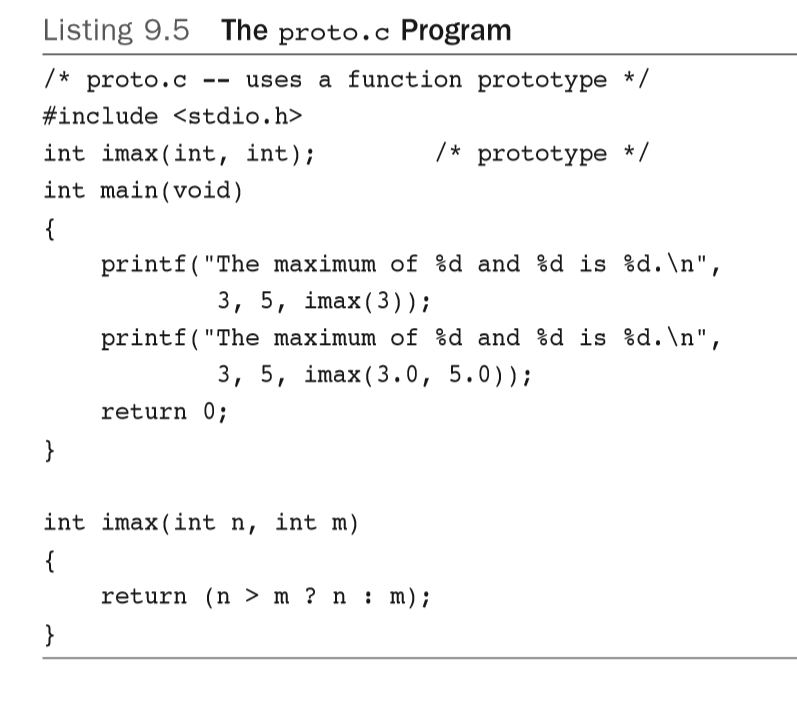
- ANSI C standard: the problems of mismatched arguments 에 대한 해결

+ function declaration에서 formal parameter의 variable types을 명시

- function prototype : return type, the number of arguments, argument types, function이름을 명시

int imax(int, int);

int imax(int a, int b);



9.2.3 No Arguments와 Unspecified Arguments

void print\_name();

* it will not check arguments.

void print\_name(void);

* function call에서 arguments를 사용하지 않는다

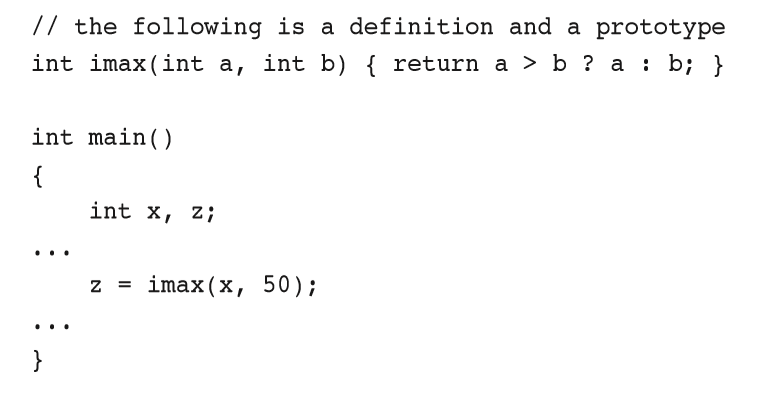
printf()와 scanf() ,

* 가변 갯수의 arguments를 사용

int printf(const char \*, ...);

9.2.4 Prototype에 함수 정의를 기술하는 경우

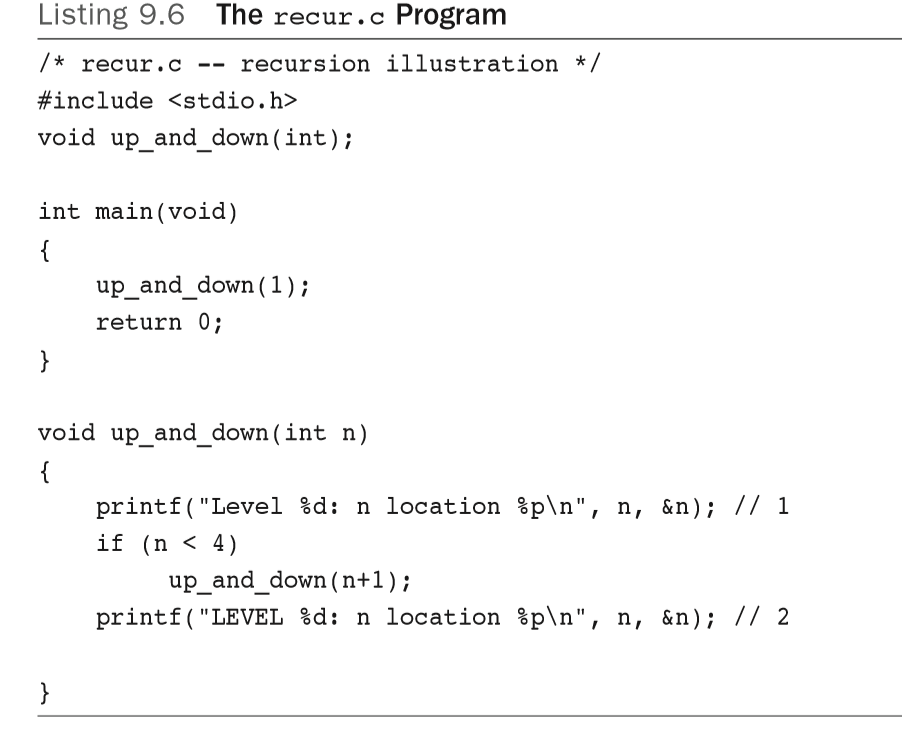
* prototype을 생략하는 방법, 그러나 prototyping 이점을 살리는 방법



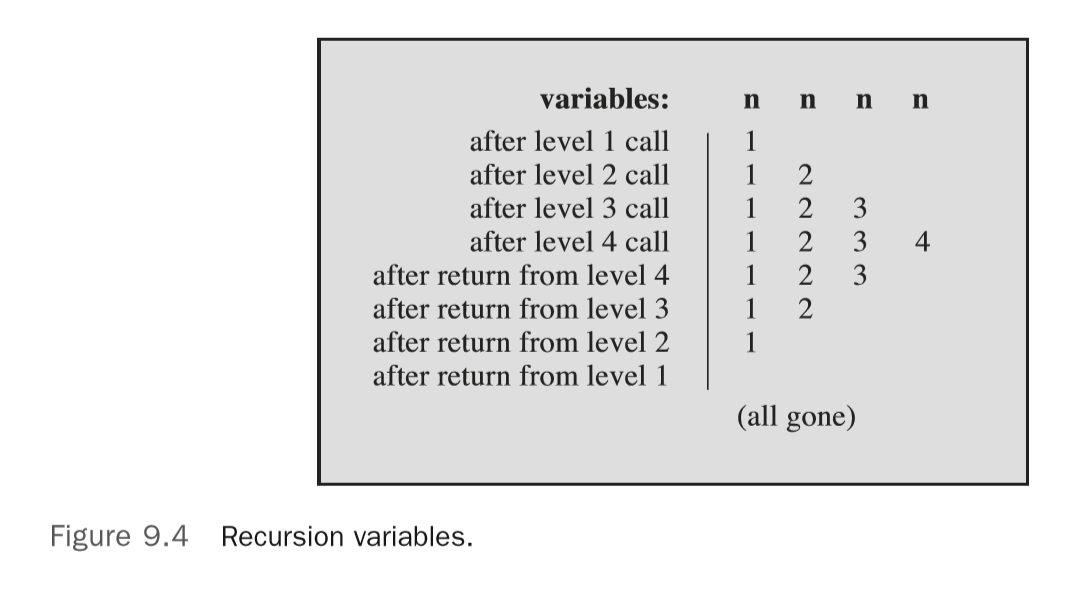
9.3 Recursion

* C에서 function이 자기 자신을 호출하는 것: recursion

9.3.1 Recursion 예제

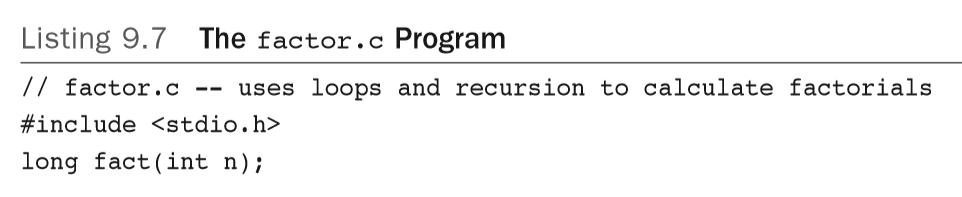


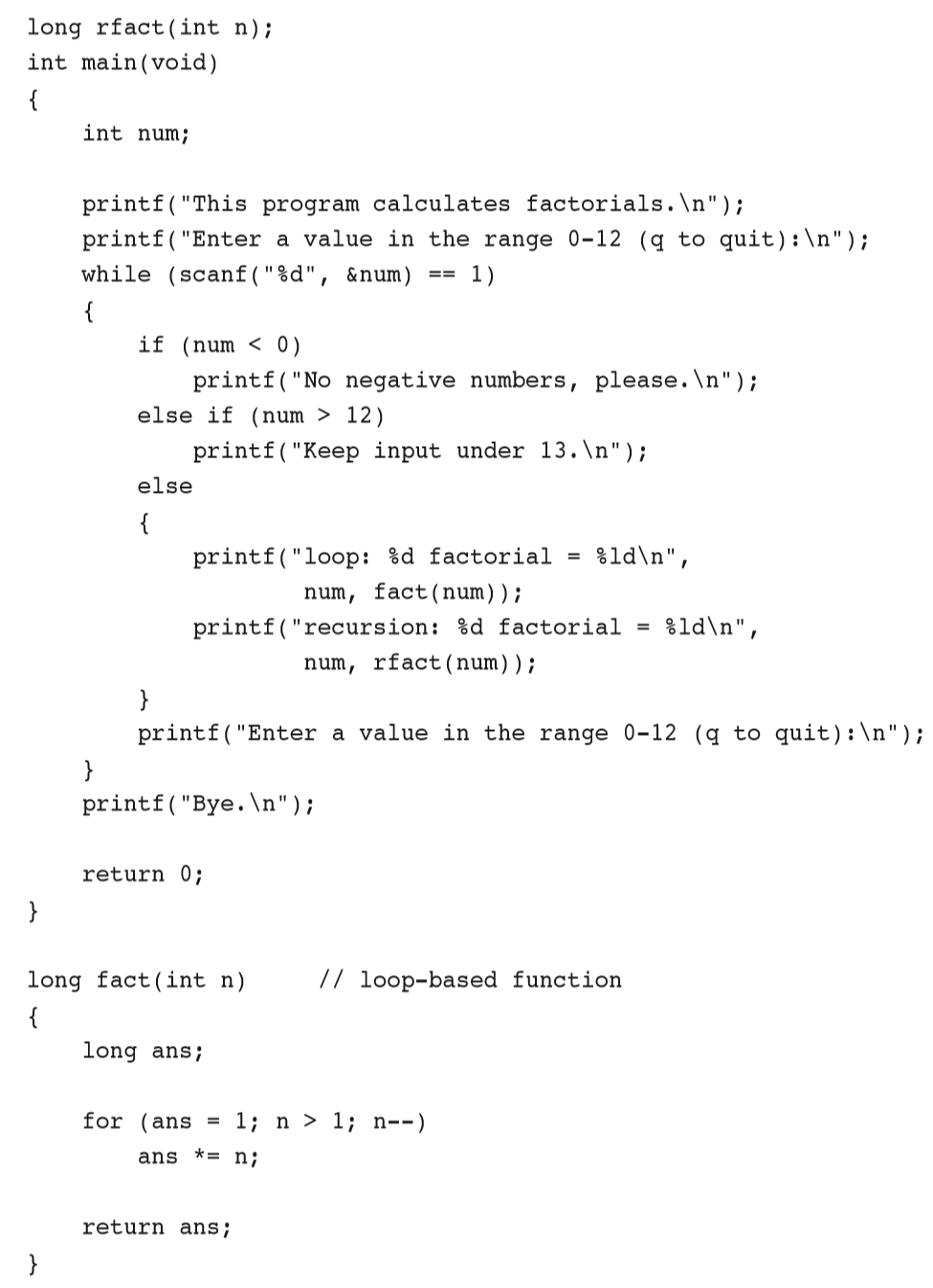
9.3.2 Recursion 실행시 변수 값 변화

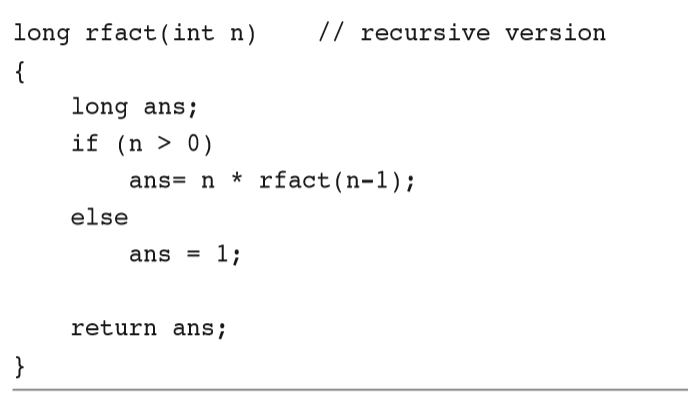


9.3.3 Tail Recursion

* recursive 호출이 function 끝 부분에서 발생: return 문 바로 앞에서 호출
  + tail recursion , 또는 end recursion: recursive call이 함수 끝 부분에서 발생.
  + -----------------------------------------------------여기까지





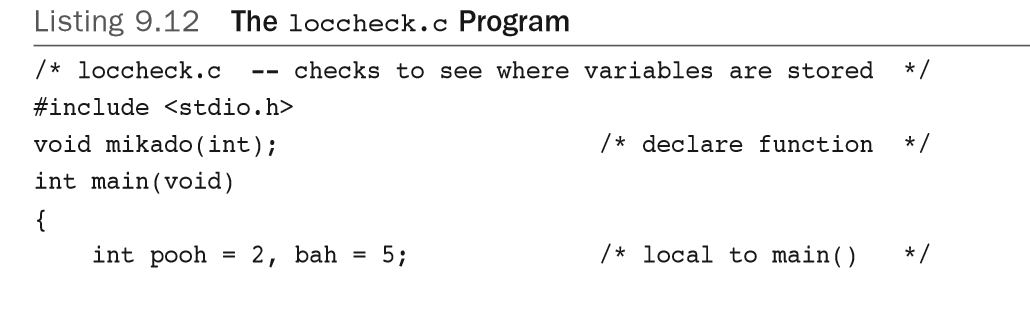


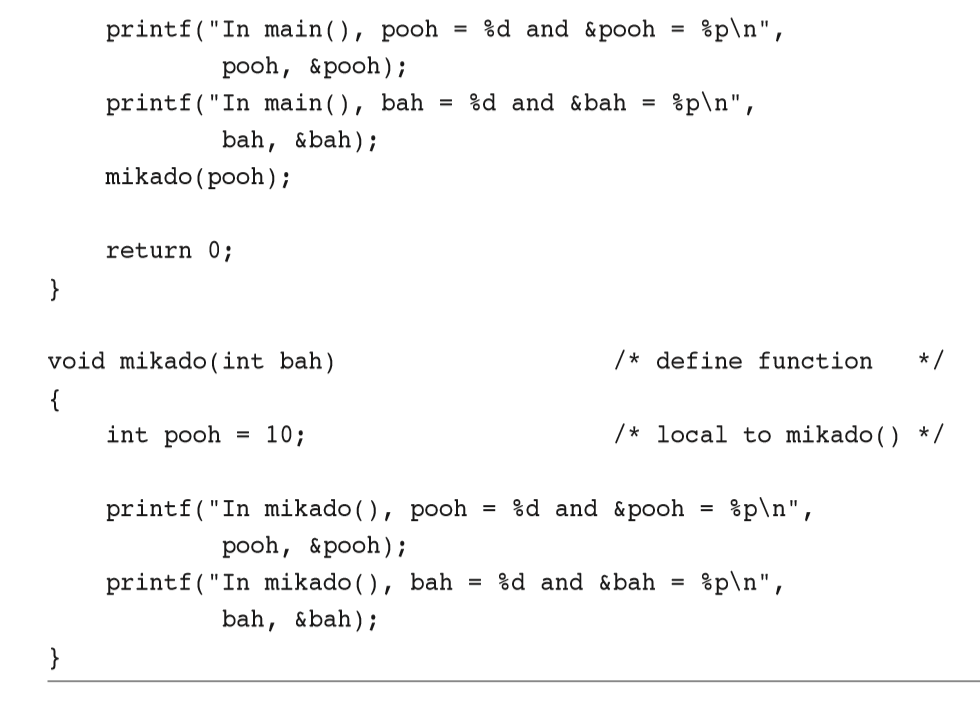
9.3.4 Recursion 찬반

* 유익한 점: recursion은 programming problems에 대한 simplest solution을 제공.
* 나쁜점: recursive algorithms은 메모리를 많이 사용할 가능성
  + recursion은 document하기도 어렵고 maintain하기도 어렵다.

9.4 Addresses를 구하는 방법: & Operator

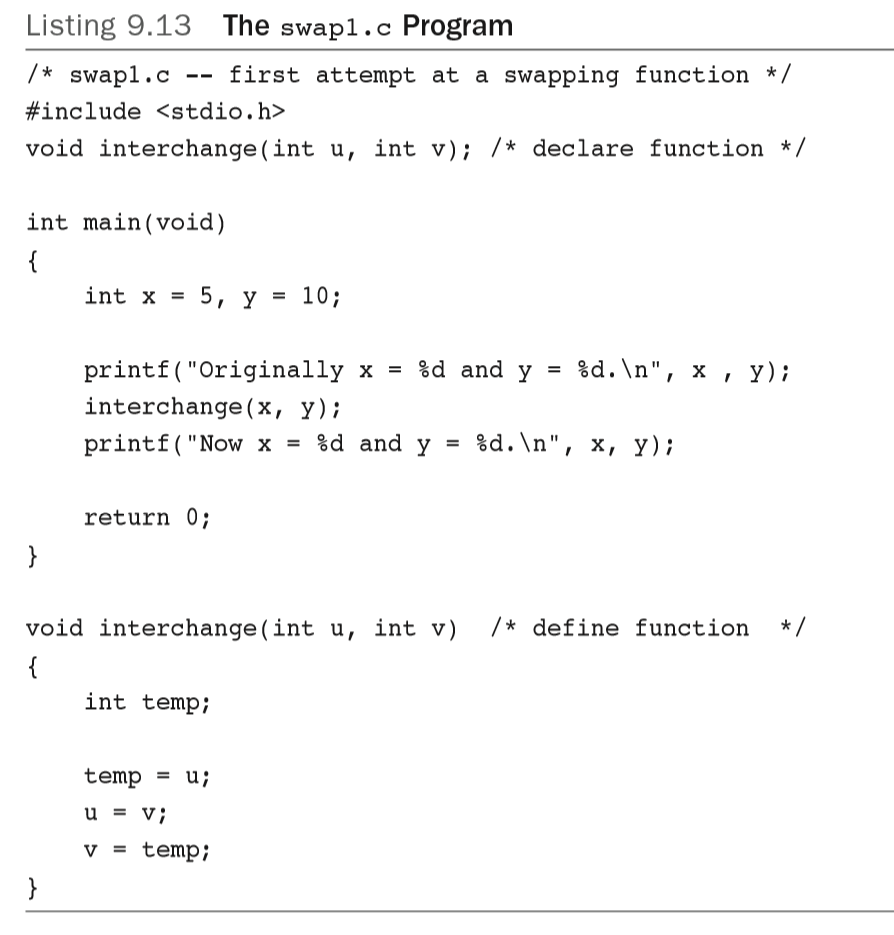
* C concepts의 가장 중요한 점: pointer = address를 저장하기 위하여 사용하는 변수
* return 값을 사용하지 않고 변수 값을 변경하는 function: address를 사용
* & operator : variable이 저장된 주소를 나타낸다





9.5 호출 함수에서 Variables 변경

* function이 variables의 값을 변경하기를 원할 때
* local variable에 대한 실습: actutal arguments와 formal parameters의 address를 출력하여 확인



9.6 Pointers: 개요

* pointer : variable (또는 더 일반적으로 얘기하면 data object을 가리킴)로서 그것의 값이 memory address

ptr = &pooh; // 변수 pooh의 address를 ptr에 치환한다

* ptr과 &pooh 차이: ptr은 variable, &pooh은 constant.
  + ptr은 modifiable lvalue, &pooh은 rvalue.

9.6.1 간접 Operator: \*

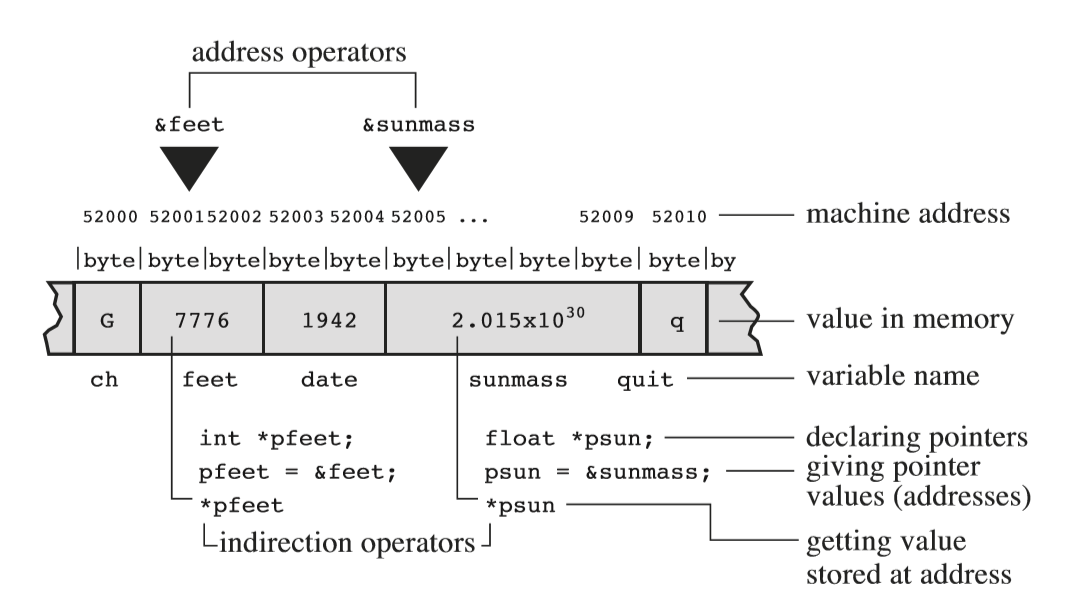
* indirection operator \* 를 사용: ptr이 가르키는 값을 가져오는 것 (dereferencing operator라고 부름)

val = \*ptr; // finding the value ptr points to

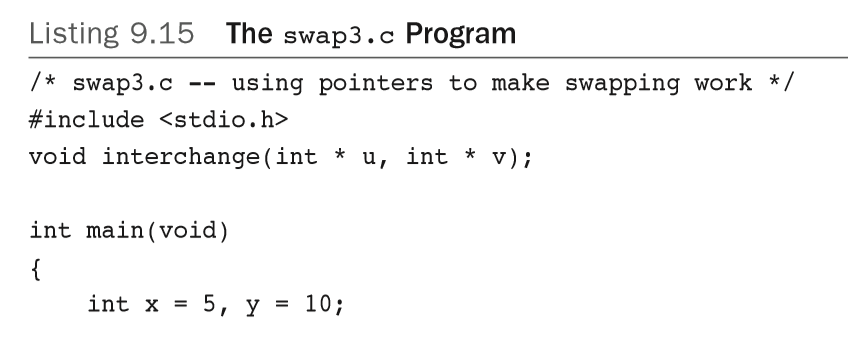
int \* pi; // pi is a pointer to an integer variable

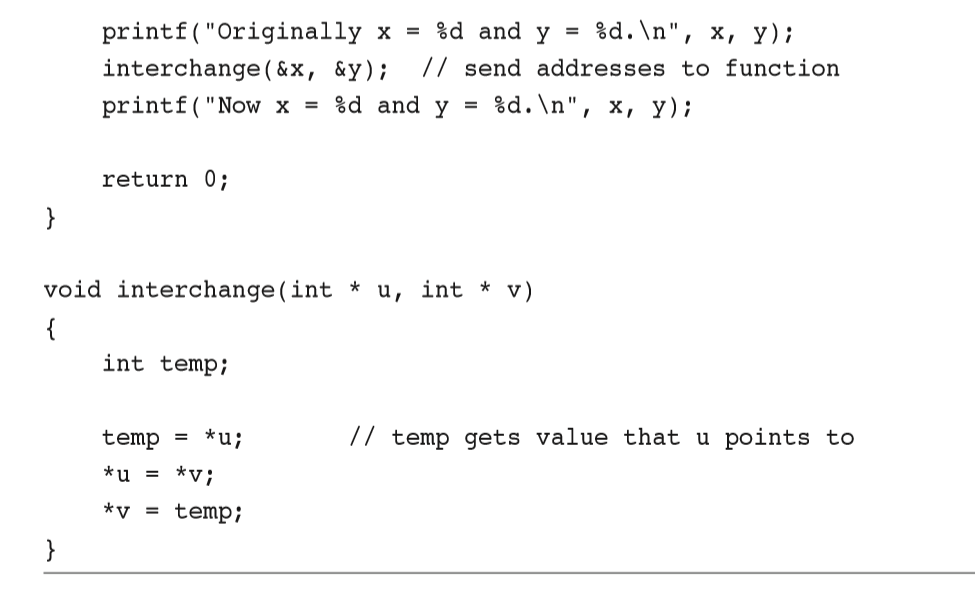
char \* pc; // pc is a pointer to a character variable

float \* pf, \* pg; // pf, pg are pointers to float variables



9.6.2 함수 간에 Pointers를 사용하여 Communicate





interchange(&x, &y);

* function은 addresses를 전달한다 .
* interchange(x, y)로 호출시 문제점을 실습

void interchange (int \* u, int \* v)

* interchange( int u, int v)로 선언시 문제점을 실습
* pointer u, v의 address를 출력하여 x,y의 address와 비교하여 실습

