5 Operators, Expressions, and Statements

■ Keyword: while , typedef

■ Operators: = - \* / % ++ -- (type)

■ C의 매우 많은 operators: 공통적인 산술 연산을 모두 포함

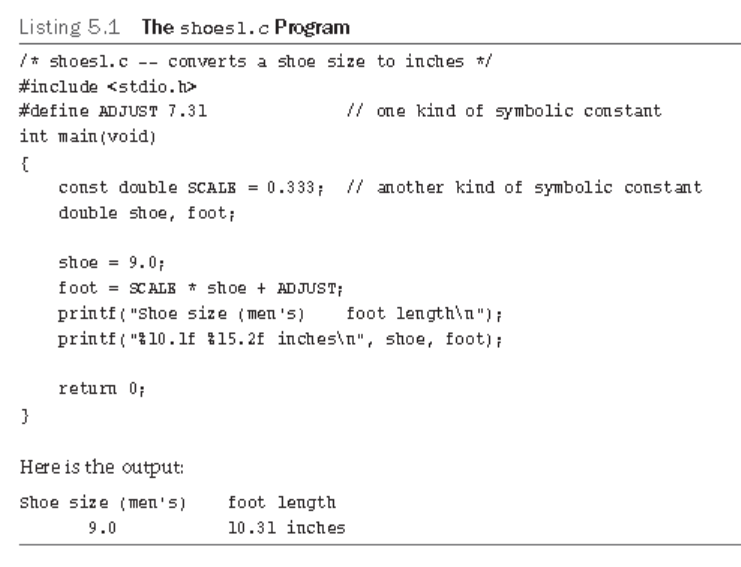
■ Operator precedence, *statement 와* *expression 용어 의미*

■ 매우 유용한 while loop

■ Compound statements, automatic type conversions, and type casts

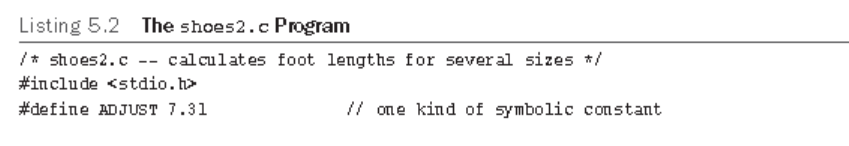
■ arguments를 사용하는 function을 작성하는 방법

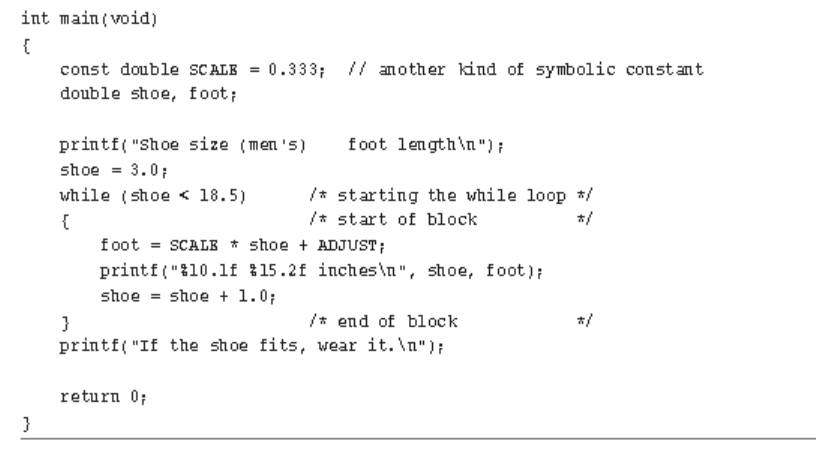
5.1 Introducing Loops



# while loop

* scanf(“%lf”, &shoe);를 수정하여 실습



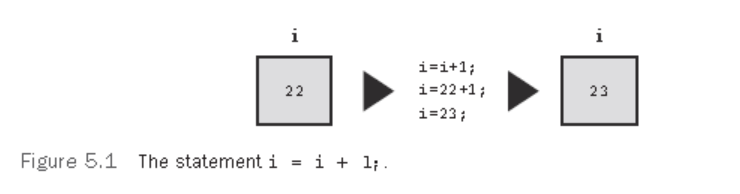


5.2 Fundamental Operators

# C는 산술 연산을 표현하기 위하여 *operators*를 사용

5.2.1 Assignment Operator: =

# 값 치환 operator



2002 = bmw; //오류 메시지 확인: lvalue

-> C가 *rvalue* 라고 부르는 것은 literal constant.

# = 기호 좌측에 있는 item은 변수 이름이어야 한다

# C는 값을 assign할 수 있는 entities를 label하기 위해 *modifiable lvalue* 용어를 사용한다

5.2.2 알아야 하는 용어: Data Objects, Lvalues, Rvalues, and Operands

# 치환문: assignment – memory location에 value를 저장

# C는 특정 data object을 식별하는 expression 또는 name을 나타내는 *lvalue* 를 사용.

* lvalue 는 object의 storage를 identify 또는 locate하는 label (Object는 actual data storage를 참조하는 것)
* lvalue는 다음 두가지를 의미:

1. object을 specify한다. memory의 address를 참조하는 것

2. 치환문의 left side 에 사용된다: “l” in lvalue.

# C 는 const modifier를 추가

* object을 생성하나 그 값은 변경할 수 없다
* lvalue이나 치환문에 사용될 수 없음

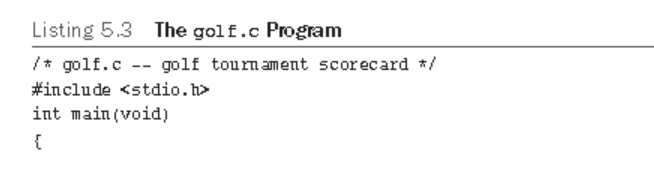
# 용어 *modifiable lvalue* 는 값을 변경할 수 있는 object을 identify

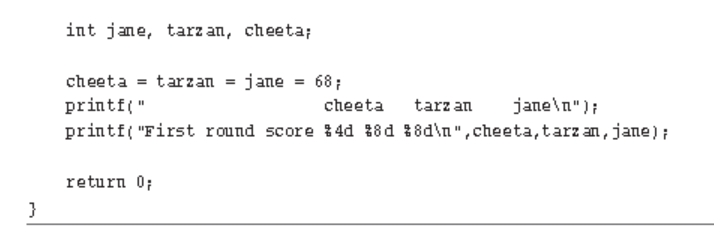
* 치환문의 좌측은 modifiable lvalue이어야 한다
* int ex; //ex는 modifiable lvalue
* const int TWO = 2; //non-modifiable lvalue
* ex = n + m; //n+m은 rvalue

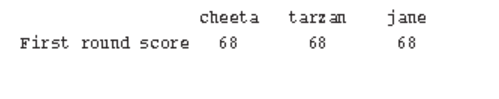
# 용어 rvalue 는 modifiable lvalue에 치환될 수 있는 quantities를 말한다

>> *r* in *rvalue* comes from *right* .

# Operands는 operators의 동작 대상







5.2.2 Addition Operator: +

5.2.3 Subtraction Operator: –

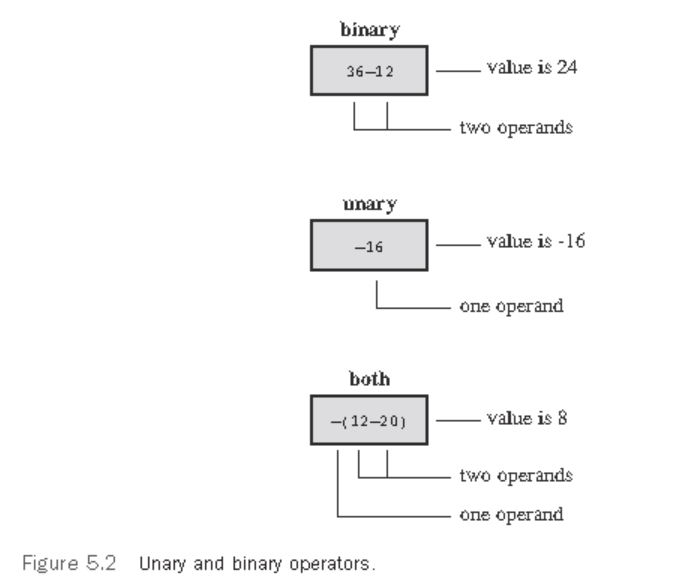
# +, – operators는 *binary* , 또는 *dyadic(2개의)* operators이라 한다

5.2.4 부호 Operators: – and +

rocky = –12;

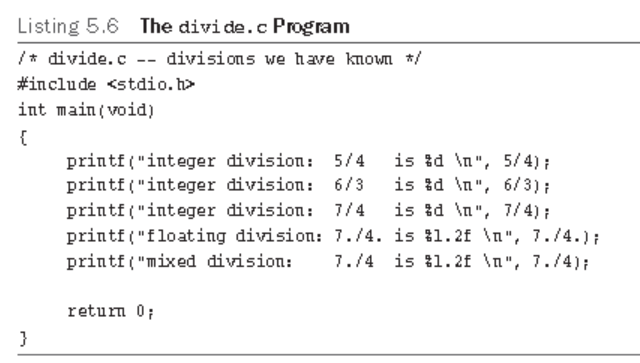
smokey = –rocky;

# *unary operator* 이라 한다



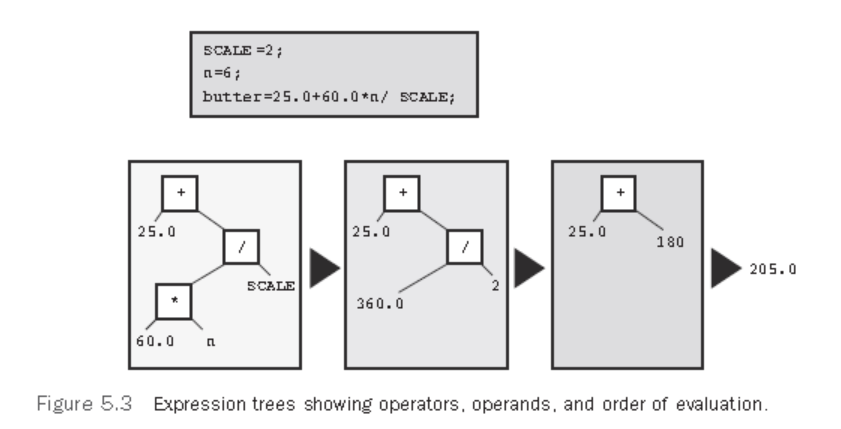
5.2.6 Division Operator: /

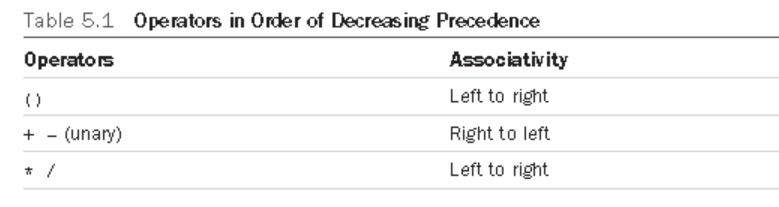
# C에서 integer division의 소숫점은 버린다: *truncation*

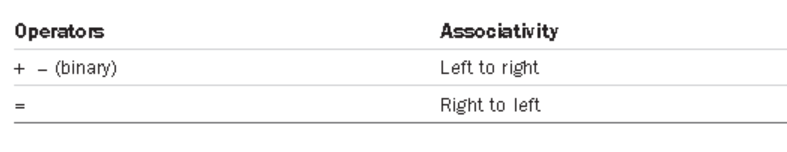


5.2.7 Operator Precedence

# 각 operator는 *precedence* level(연산자 우선순위)을 갖는다

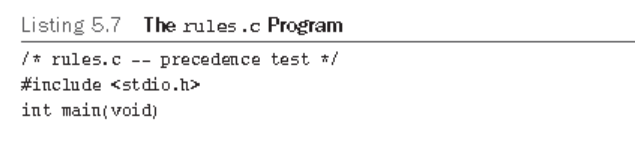


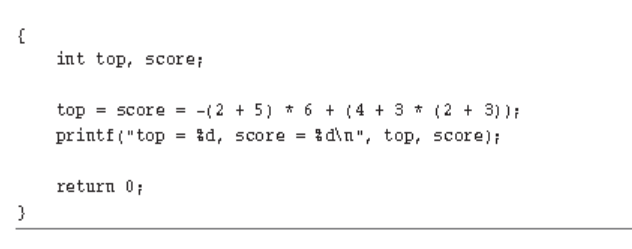




5.2.8 Precedence와 Evaluation order

# expression에서 evaluation order를 결정하는 규칙



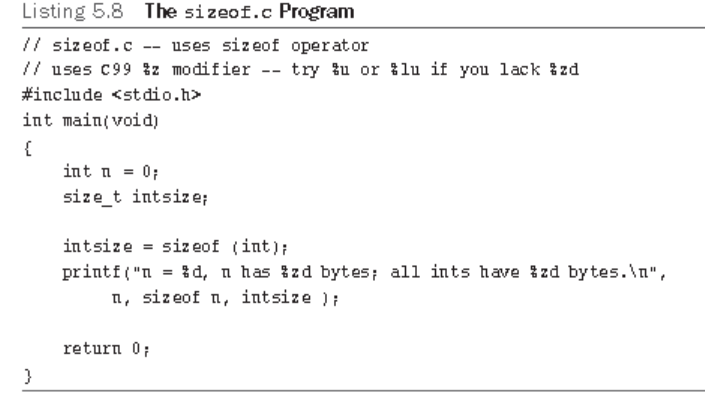


5.3 추가 Operators

5.3.1 sizeof Operator와 size\_t Type

# sizeof operator는 크기를 return한다: operand의 크기를 byte로 return

* operand가 type(float)이면 괄호사용: sizeof(int)
* operand가 특정 variable(array)이면 괄호 사용하지 않아도 됨: sizeof foo



# sizeof operator는 type size\_t의 값을 return

* unsigned integer type: %zd

# C는 typedef 정의 기능이 있다

기존 type에 대한 alias를 제공

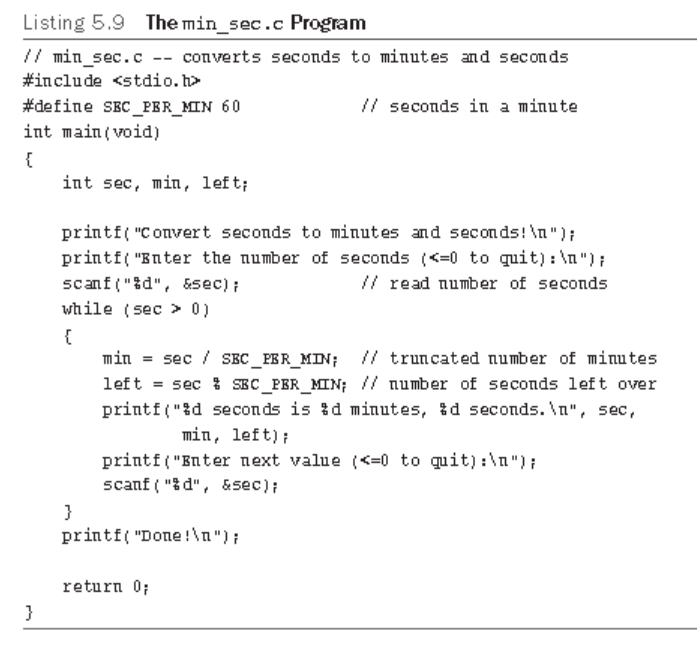
typedef double real;

real deal; // using a typedef

* C header files은 typedef를 사용하여 unsigned int 대신에 size\_t를 정의

5.3.2 Modulus Operator: %

# *remainder(*나머지)를 제공



- a,b가 정수이면 a%b = a - (a/b)\*b;// a = -11, b = 5

-11 - (-11/5) \* 5 = -11 -(-2)\*5 = -11 -(-10) = -1

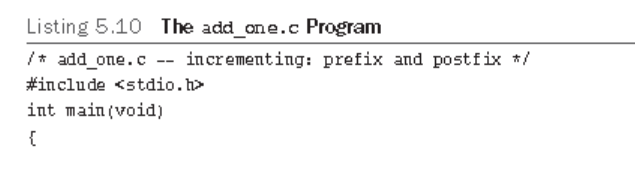
5.3.3 Increment 와 Decrement Operators: ++, --

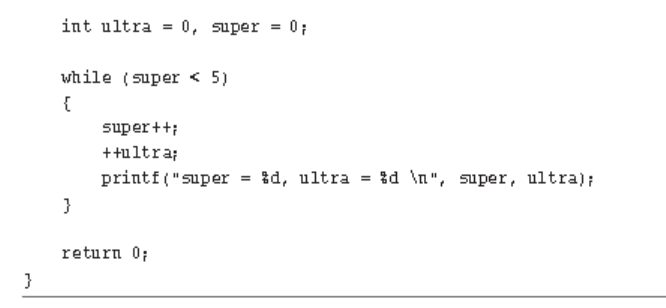
# *prefix* mode: ++I;

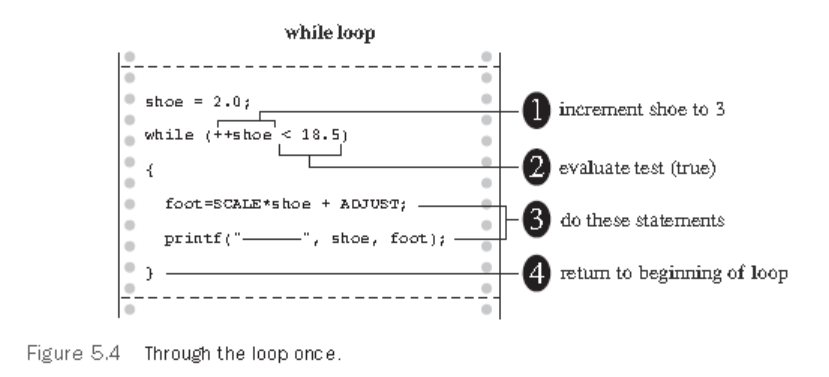
* increment n; then use it

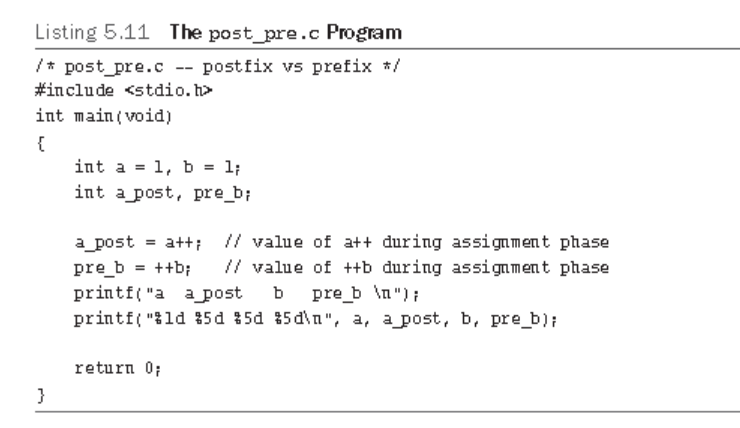
# *postfix* mode: i++;

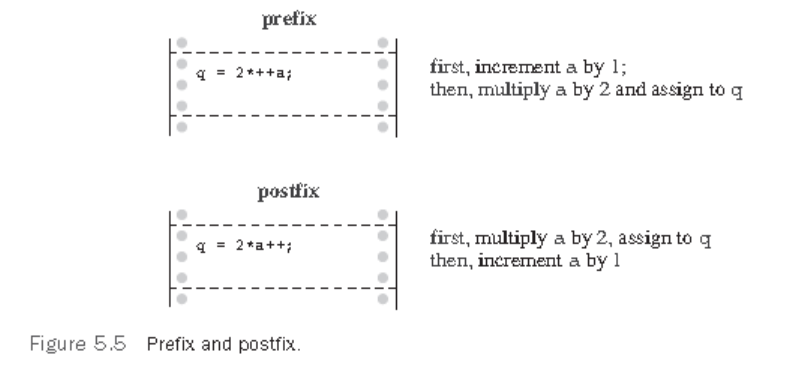
* use n; then increment it

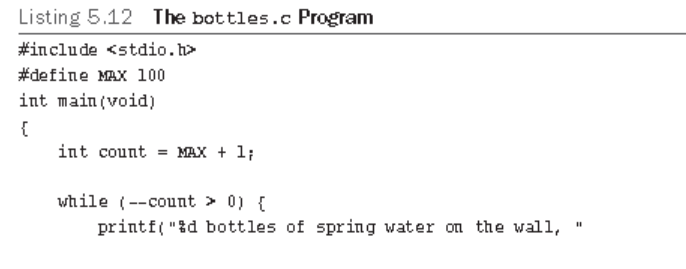


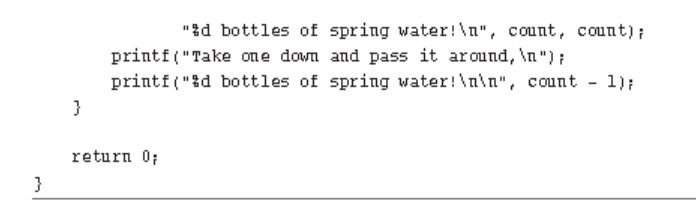












# > operator는 “is greater than”을 나타냄. < (“is less than”) : *relational operator* .

5.3.4 Precedence

# 증감 operator는 메우 높은 precedence을 가짐

* **증감 operator는 오직 lvalue**에만 적용됨

x\*y++ means (x)\*(y++)

* (x\*y)++ : wrong //x\*y는 rvalue이다

n = x\*y++; //n은 x\*y 값을 갖는다

n = x\*++y; //n은 증가된 값의 곱셈

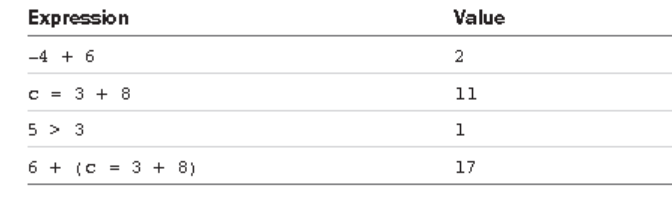
5.4 Expressions과 Statements

5.4.1 Expressions

# *expression* 은 operator와 operand의 결합

# operand는 operator가 적용하는 대상

# 모든 Expression은 값을 갖는다

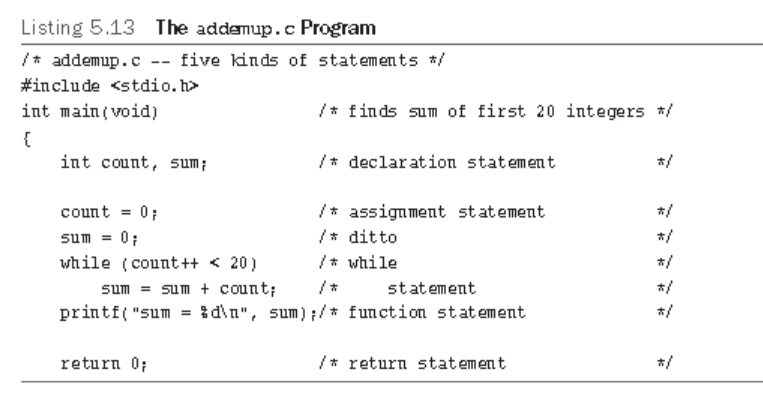


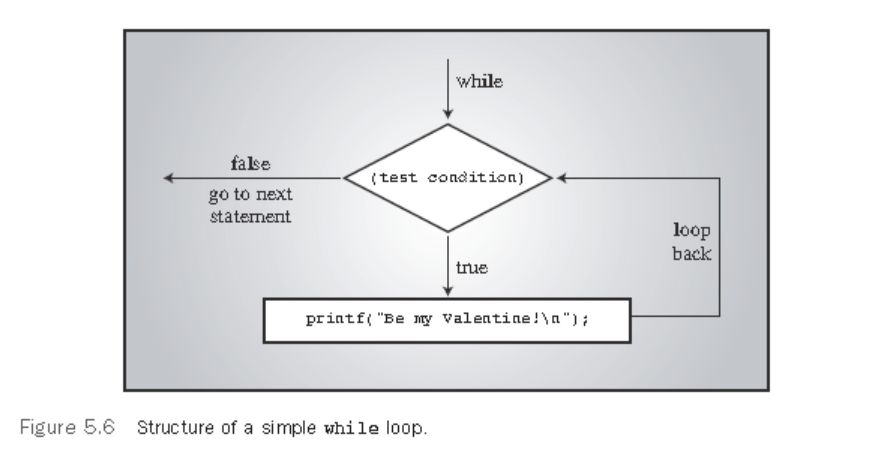
5.4.2 Statements

# *Statements*는 프로그램의 primary building blocks이다

# *program* 은 일련의 statements(punctuation: 세미콜론을 포함).

* 선언 statement: int toes;
* 치환 statement: toes = 12;
* 함수 호출 statement: printf("%d\n", toes);
* 구조적 statement: while (toes < 20) toes = toes + 2;
* Return statement: return 0;
* null statement: ; /\* does nothing \*/





* while statement는 *structured statements*

5.4.3 Side Effects와 Sequence Points

# *side effect* 는 data object이 변경되는 것이다

states = 50;

* 변수 states가 50이 된다 . Side effect?
* C program의 주요 의도는 expression을 evaluate하는 것이다

4+6; => C 는 evaluate하여 10이 결과이다

states = 50; => C는 evaluate하여 결과가 50이다

* expression의 evaluate는 states 변수가 50이 되게 하는 side effect가 있다 .

# 증감 operators, 치환 operators도 side effects를 갖는다

# printf() 함수를 호출할 때, 값을 display하는 것은 side effect이다.

* printf()의 evaluate하는 것은 dispaly하는 item의 숫자이다

# *sequence point* 는 다음 step으로 가기 전에 모든 side effects 가 evaluate되는 program execution시의 한 point를 말한다

* C에서 statement의 semicolon은 sequence point를 나타낸다

# *full expression*?

* larger expression 의 subexpression이 아닌 것

y = (4+x++)+(6+x++); //4+x++는 full expression이 아니다

// y = (4+x++)+(6+x++); 전체는 full expression이다. semicolon 은 sequence point를 나타낸다

while (guests++ < 10) printf("%d \n", guests);//guests++<10은 full expression이다

* + - a while loop test condition은 sequence point이다
    - C는 side effect (incrementing guests )가 printf() 실행 전에 완료되는 것을 보장한다

5.4.4 Compound Statements (Blocks)

# *compound statement* 는 braces로 둘러쌓인 두개 이상의 statements

* *block*

/\* fragment 1 \*/

index = 0;

while (index++ < 10)

sam = 10 \* index + 2;

printf("sam = %d\n", sam);

/\* fragment 2 \*/

index = 0;

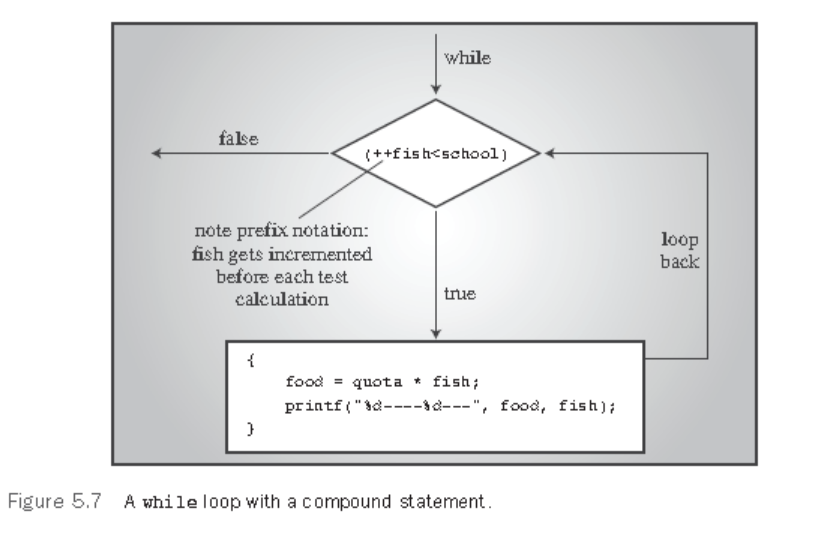
while (index++ < 10)

{

sam = 10 \* index + 2;

printf("sam = %d\n", sam);

}



5.5 Type Conversions

# type conversions을 자동으로 처리는 규칙

1) float는 자동으로 double로 변환

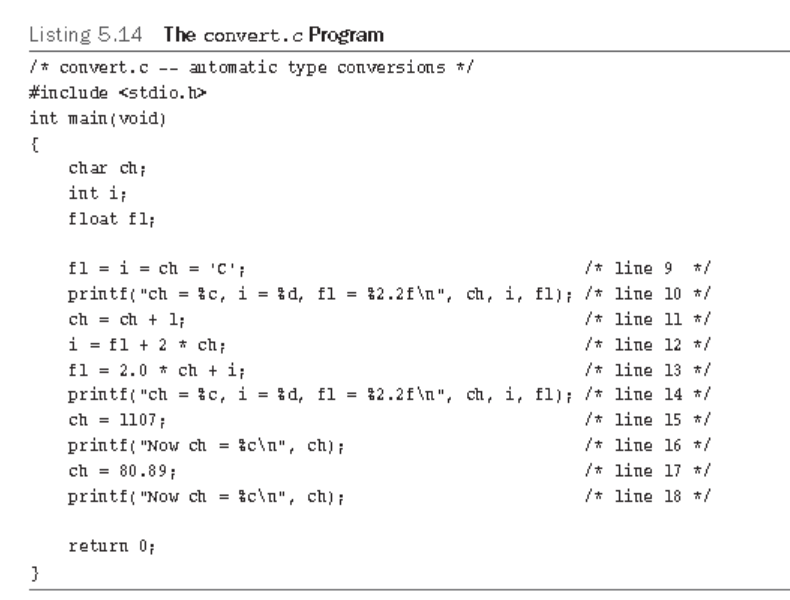
- larger types으로 conversion이므로 *promotions* .

2) 두가지 type이 섞여있을 때는 두가지 type의 higher ranking 으로변환

3) highest ranking of types: **long double , double , float , unsigned long long , long long , unsigned long , long , unsigned int , and int** .

*4) demotion* : 변환 값이 lower-ranking type으로 바뀌는 것

*5)* function arguments로 전달될 때, char/short은 int로 , float는 double로 변환되는 것을 automatic promotion이라 한다.function prototyping에 의해 override된다고 한다



5.5.1 The Cast Operator

# 인위적인 type conversion.

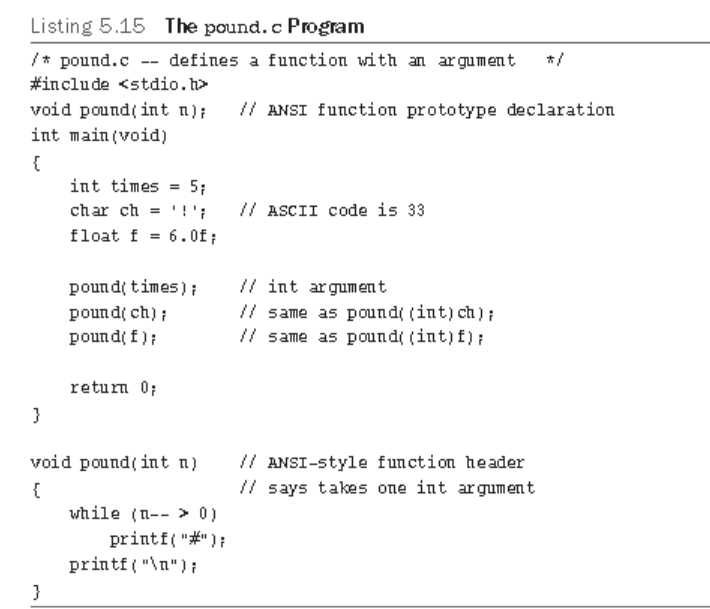
* *cast* : 괄호 안에 변환하고자 하는 type을 기술
* (type): *cast operator* .

mice = 1.6 + 1.7;

mice = (int) 1.6 + (int) 1.7;

5.6 Function with Arguments

# arguments를 사용하는 function 작성



void pound(int n)

* function은 type int argument를 사용

void fcn(void);

# function에서 argument 선언을 *formal argument* 또는 *formal parameter* .

* formal parameter는 int 변수

# function call은 값을 *passes* 하고, 전달된 값을 *actual argument* 또는 *actual parameter*

* function call pound(10)은 actual argument 10을 해당 함수에 passes

# 전달된 variable names은 해당 function에 local variable

* function에 전달된 변수는 main() 함수의 변수와 다르다(이름이 같아도)
* pound(times) 전후의 변수 times 값을 출력하여 확인

# *prototype* : function declaration으로 function’s return value과 arguments을 기술.

■ function이 return value (void 는 return 값이 없음을 표시).

■ function이 1개의 argument을 갖는 경우: type int value.

void pound(); /\* pre-ANSI function declaration \*/ 실습 대상

pound ((int) f); // force correct type, 실습 대상

- function call에서 explicit type cast를 사용

5.7 예제 Program

