2010-1학기 프로그래밍입문(1)

4장 제어의 흐름

박 종 혁

Tel: 970-6702
Email: jhpark1@snut.ac.kr

목 차

- 4.1 관계, 등가, 논리 연산자
- 4.2 관계 연산자와 수식
- 4.3 등가 연산자와 수식
- 4.4 논리 연산자와 수식
- 4.5 복합문
- 4.6 수식과 공백 문장
- 4.7 if와 if-else 문
- 4.8 while 문
- 4.9 for 문
- 4.10 예제: 부울 변수
- 4.11 콤마 연산자
- 4.12 do 문
- 4.13 예제: 피보나치 수열
- 4.14 goto 문
- 4.15 break 문과 continue 문
- 4.16 switch 문
- 4.17 조건부 연산자

제어의 흐름

- 제어의 순차적 흐름 - 프로그램의 문장은 일반적으로 순차적으로 실행됨
- 동작의 선택, 반복을 위해 순차적 흐름을 변경할 필요가 있음
  - 선택 : if, if-else, switch
  - 반복 : while, for, do
  - 이러한 구문에 관계, 등가, 논리 연산자가 사용됨

관계, 등가, 논리 연산자

- 관계 연산자
  - ~보다 작다 : <
  - ~보다 크다 : >
  - ~보다 작거나 같다 : <=
  - ~보다 크거나 같다 : >=

관계, 등가, 논리 연산자

- 등가 연산자
  - 같다 : ==
  - 같지 않다 : !=
- 논리 연산자
  - (단항)부정 : !
  - 논리곱 : &&
  - 논리합 : ||

참과 거짓

- C에서 거짓은 0 값으로 나타내고, 참은 0 아닌 다른 값으로 나타냄
- 거짓의 값은 임의의 0 값이 될 수 있음
  - 0, 0.0, 널 문자 '\0', NULL 포인터 값
- 참 값은 임의의 0이 아닌 값
- a < b와 같은 수식은 참이나 거짓의 값을 가짐
  - 이 수식은 참이면 정수 값 1을, 거짓이면 정수 값 0을 생성

관계 연산자

- 다음 관계 연산자는 모두 이항 연산자임

  < > <= >=

- 관계 수식의 값

a - b	a < b	a > b	a <= b	a >= b
양수	0	1	0	1
영	0	0	1	1
음수	1	0	1	0

관계 수식 예제

선언 및 초기화

```
char     c = 'w';
int      i = 1, j = 2, k = -7;
double   x = 7e+33, y = 0.001;
```

수식	동일한 수식	결과 값
`'a' + 1 < c`	`('a' + 1) < c`	1
`- i - 5 * j >= k + 1`	`((-i) - (5 * j)) >= (k + 1)`	0
`3 < j < 5`	`(3 < j) < 5`	1
`x -3.333 <= x + y`	`(x - 3.333) <= (x + y)`	1
`x < x + y`	`x < (x + y)`	0

▶ j =7;

```
printf ("%d₩n" , 3 < j < 5);          /* 1 gets printed, not 0 */
```

▶ 3 < j 와 j < 5 를 검사하는 올바른 방법은 3 < j && j <5

▶ x < x + y

((x - (x + y)) < 0.0

등가 연산자

- 등가 수식의 값

값 ex1 - ex2	ex1 == ex2	ex1 != ex2
zero	1	0
nonzero	0	1

- a != b와 !(a == b)는 동일한 수식
- (주의) a == b와 a = b는 유사하지만, 완전히 다른 수식임

```
if (a = 1)
    ......
```

등가 수식 예제

선언 및 초기화		
`int i = 1, j = 2, k = 3;`		
수식	동일한 수식	결과 값
`i == j`	`j == i`	0
`i != j`	`j != i`	1
`i + j + k == -2 * -k`	`((i + j) + k) == ((-2) * (-k))`	1

▶ 잘못된 사용 예

```
a = b          /* an assignment statement */
a = = b - 1    /* space not allowed */
(x + y) =! 44  /* syntax error: equivalent to (x + y) = (!44) */
```

논리 연산자

- 논리 연산자
  - ! : 단항 연산자
  - &&, || : 이항 연산자
- ! 연산자

값 ex1	!ex1
zero	1
nonzero	0

! 연산자

선언 및 초기화

```
char   c = 'A';
int    i = 7, j = 7;
double    x = 0.0,  y = 2.3;
```

수식	동일한 수식	결 과 값
`! c`	`! c`	0
`! (i - j)`	`! (i - j)`	1
`! i - j`	`(! i) - j`	-7
`! ! (x + y)`	`! (! (x + y))`	1
`! x * ! ! y`	`(! x) * (!(! y))`	1

&&, || 연산자

- 수식의 값

값 ex1	ex2	ex1 && ex2	ex1 || ex2
zero	zero	0	0
zero	nonzero	0	1
nonzero	zero	0	1
nonzero	nonzero	1	1

&&, || 연산자

선언 및 초기화

```
char    c = 'B';
int     i = 3, j = 3,  k = 3;
double    x = 0.0,  y = 2.3;
```

수식	동일한 수식	결과값
`i && j && k`	`(i && j) && k`	1
`x || i && j - 3`	`x || (i && (j - 3))`	0
`i < j && x < y`	`(i < j) && (x < y)`	0
`i < j || x < y`	`(i < j) || (x < y)`	1
`'A' <= c && c <= 'Z'`	`('A' <= c) && (c <= 'Z')`	1
`c -1 == 'A' || c + 1 == 'Z'`	`((c - 1) == 'A') || ((c + 1) == 'Z')`	1

▶ 잘못 사용된 예

```
a!              /* out of order */
a != b          /* != is the token for the "not equal" operator */

a &&             /* one operand missing */
a l l b         /* extra space not allowed */
a & b           /* this is a bitwise operation */
&b              /* the address of b */
```

단축 평가

- 결과가 참인지 거짓인지 판명되면 더 이상 다음 수식을 평가하지 않음
- expr1 && expr2
  - expr1이 **거짓**일 때,
    이 수식은 이미 거짓이라는 것이 판명되고
    따라서 expr2는 평가되지 않음
- expr1 || expr2
  - expr1이 **참**일 때,
    이 수식은 이미 참이라는 것이 판명되고
    따라서 expr2는 평가되지 않음

복합문

- 중괄호 {}로 묶여진 선언문과 문장
- 문장들을 실행 가능한 하나의 단위로 그룹화함
- 선언문이 복합문의 시작 부분에 있으면, 그 복합문을 블록이라고 함
- 복합문 자체도 하나의 문장임
  - C에서 문법적으로 한 문장이 들어가는 자리에 복합문을 사용할 수 있음

▶ 복합문의 예

```
{
a = 1;
 {
   b = 2;
   c = 3;
 }
}
```

수식과 공백 문장

- 수식 문장은 수식 다음에 세미콜론이 붙은 것
- 세미콜론은 수식을 문장으로 만듦
- 각 문장은 다음 문장으로 넘어가기 전에 완전히 평가됨
- 공백 문장은 하나의 세미콜론으로 작성됨

수식과 공백 문장

- 공백 문장은 구문상으로는 문장이 필요하지만, 의미상으로 그 문장이 필요 없을 때 유용

- 예

```
a = b;              /* an assignment statement */
a + b + c;
    /* legal, but no useful work gets done */
;                       /* an empty statement */
printf("%d\n", a);      /* a function call */
```

if

- if 문의 형식

  if (expr)

  statement

  - expr이 0이 아니면(참), statement가 실행됨
  - 0 이면, statement는 실행되지 않고 다음 문장으로 제어가 넘어감

if

- 예제

```
if (y != 0.0)
    x /= y;
if (c == ' ') {
    ++blank_cnt;
    printf("found another blank\n");
}
```

if-else 문

- if-else 문의 형식

```
if (expr)
    statement1
else
    statement2
```

- expr이 0이 아니면(참), statement1이 실행되고,
- 0 이면, statement2가 실행됨

if-else 문

- 예제

```
if (x < y)
    min = x;
else
    min = y;
```

if, if-else 예제

- 예제 1

```
if (c >= 'a' && c <= 'z')
    ++lc_cnt;
else {
    ++other_cnt;
    printf("%c is not a
                lowercase letter", c);
}
```

if, if-else 예제

- 예제 2

```
if (i != j) {
    i += 1;
    j += 2;
} ;
else
    i -= j;      /* syntax error */
```

if, if-else 예제

- 예제 3

```
if (a == 1)
    if (b == 2)
        printf("***\n");
```

- 예제 4

```
if (a == 1)
    if (b == 2)
        printf("***\n");
    else
        printf("###\n");
```

```
if (a == 1)
    if (b == 2)
        printf("***\n");
else
    printf("###\n");
```

while 문

- 일반적인 형태

  while (expr)

    statement

  next statement

- 동작단계
  - expr이 0이 아니면(참), statement가 실행되고 제어는 다시 while 루프의 시작 부분으로 다시 전달됨
  - expr이 0(거짓)이면, 제어는 next statement로 넘어감 (따라서 while 루프의 몸체는 0번 이상 수행됨)

while 문 예제

- 옳은 예제

```
while ((c = getchar()) != EOF) {
    if (c >= 'a' && c <= 'z')
        ++lowercase_letter_cnt;
    ++total_cnt;
}
```

- 잘못된 예제

```
while (++i < LIMIT) do { // do is not allowed
    j = 2 * i + 3;
    printf("%d\n", j);
}
```

- 바람직하지 않은 코드

```
printf("Input an integer :     ");
scanf("%d",  &n);
while (--n)
    .....          /* do something */
```

- 바람직한 코드

```
printf("Input an integer :     ");
scanf("%d",  &n);
while (--n > 0)
    .....          /* do something */
```

■ 문자 종류별로 사용된 문자의 수를 세는 프로그램.

```
/* count blanks, digits, letters, newlines, and others. */

#include <stdio.h>

int main(void)
{
    int blank_cnt = 0, c, digit_cnt = 0,
    letter_cnt = 0, nl_cnt = 0, other_cnt = 0;
```

```
while ((c = getchar()) != EOF) /* braces not necessary */
        if (c == ' ')
        + + blank_cnt;
        else if (c >= '0' && c <= '9')
            + + digit_cnt;
        else if (c >= 'a' && c <= 'z' l l c >= 'A' && c <= 'Z')
            + + letter_cnt;
        else if (c == '₩n')
            + + nl_cnt;
        else
            + + other_cnt;
```

```
    printf("%10s%10s%10s%10s%10s%10s₩n₩n",
            "blanks", "digits", "letters", "lines", "others", "total");
    printf("%10d%10d%10d%10d%10d%10d₩n₩n",
            blank_cnt, digit_cnt, letter_cnt, nl_cnt, other_cnt,
            blank_cnt + digit_cnt + letter_cnt + nl_cnt + other_cnt);
    return 0;
}
```

■ cnt_char < cnt_char.c

화면에 나타나는 출력은 다음과 같다.

```
blanks  digits   letters lines    others  total
197     31       348     27       180     783
```

for 문

- 일반적인 형태

```
for (expr1; expr2; expr3)
   statement
next statement
```

- 동작 단계
  1. expr1 평가 /* 보통 expr1은 초기화를 위해 사용 */
  2. expr2 평가 /* 보통 expr2는 제어 수단으로 사용 */
  3. expr2 평가 결과가 0이 아니면,
     1) statement 실행
     2) expr3 실행 /* 보통 expr3은 제어 값을 조정하기 위해 사용 */
     3) 2 단계부터 다시 실행
  4. expr2 평가 결과가 0이면,
     1) next statement 실행

- for의 일반적인 형태에서 expr2를 사용하고 statement가 continue를 포함하고 있지 않다면, 다음과 같은 의미임

```
expr1;
while (expr2) {
    statement
    expr3;
}
next statement
```

for 문 예제

```
for (i = 1; i <= n; ++i)
   factorial *= i;

for ( ; i <= 10; ++i)
   sum += i;

for ( ; i <= 10; )
   sum += i++;

for ( ; ; ) {
   sum += i++;
   printf("%d\n", sum);
}
```

▶ 예

```
for (i =1; i <=n; + + i)
factorial *= i;

for (j = 2; k % j == 0; + + j) {
        printf("%d is a divisor of %d₩n", j, k);
        sum + = j;
}
```

▶ 잘못된 사용 예

```
for (i = 0, < n, i + = 3) /*semicolons are needed */
        sum + = i;
```

for 문 예제

■ 1에서 10 까지 정수의 합을 계산.

```
--> i = 1;
        sum = 0;
        for ( ; i <= 10; + + i)
          sum + = i;
--> i = 1;
        sum = 0;
        for ( ; i <= 10 ; )
          sum + = I+ + ;
```

■ i = 1;

```
sum = 0;
for ( ; ; ) {
        sum + = i+ + ;
        printf("%d₩n", sum);
}
```

예제 : 부울변수

■ /* Print a table of values for some boolean functions. */

```
#include <stdio.h>

int main(void)
{
        int b1, b2, b3, b4, b5;             /* boolean variables */
        int cnt = 0;

printf("₩n%5s%5s%5s%5s%5s%5s%7s%7s%11s₩n₩n", /*headings */
        "cnt", "b1", "b2", "b3", "b4", "b5",
        "fct1", "fct2", "majority");
```

```
    for (b1 = 0; b1 <= 1; ++b1)
        for (b2 = 0; b2 <= 1; ++b2)
            for (b3 = 0; b3 <= 1; ++b3)
                for (b4 = 0; b4 <= 1; ++b4)
                    for (b5 = 0; b5 <= 1; ++b5)
                        printf("%5d%5d%5d%5d%5d%5d%6d%7d%9d₩n",
                            ++cnt, b1, b2, b3, b4, b5,
                             b1 || b3 || b5, b1 && b2 || b4 && b5,
                             b1 + b2 + b3 + b4 + b5 >= 3);
    putchar('₩n');
    return 0;
}
```

==> 출력

cnt	b1	b2	b3	b4	b5	fct1	fct2	majority
1	0	0	0	0	0	0	0	0
2	0	0	0	0	1	1	0	0
3	0	0	0	1	0	0	0	0

.....

콤마 연산자

- C의 모든 연산자들 중에서 가장 낮은 우선순위를 가짐
- 피연산자로 수식을 갖는 이항 연산자
- 좌에서 우로의 결합 법칙
- 콤마 수식의 값은 콤마 연산자의 오른쪽 수식의 값과 형이 됨
- for 문에서 많이 사용

```
for (sum = 0, i = 1; i <= n; ++i)
    sum += i;
```

선언 및 초기화

```
int      i, j, k = 3;
double   x = 3.3;
```

수식	동일 수식	값
`i = 1, j = 2, ++ k + 1`	`((i = 1), (j = 2)), ((++k) + 1)`	5
`k != 1, ++ x * 2.0 + 1`	`(k != 1), (((++ x) * 2.0) + 1)`	9.6

do 문

- 일반적인 형태

```
do
   statement
while (expr);
next statement
```

- 동작 단계
  1. statement 실행
  2. expr 평가
  3. expr 평가 결과가 0이 아니면,
     1) 1 단계부터 다시 실행
  4. expr 평가 결과가 0이면,
     1) next statement 실행

- do 문은 몸체 부분을 반드시 한번 이상 실행해야 할 때 유용
- 예제

```
i = 0;
sum = 0;
do {
    sum += i;
    scanf("%d", &i);
} while (i > 0);
```

do 문 예

- 양의 정수만을 읽는 프로그램

```
do {
        printf("Input a positive integer: ");
        scanf("%d", &n);
        if (error = (n <= 0))
            printf("₩nERROR: Do it again!₩n₩n");
        } whole (error);
```

- float 형이나 double 형의 식에 대한 등가 검사는 컴퓨터에서 수 표현의 정확도 때문에 의도대로 작동되지 않을 수 있음

루프 문을 사용할 때 주의사항

- 예제

```
#include <stdio.h>
int main(void){
   int      cnt = 0;
   double   sum = 0.0, x;
   for (x = 0.0;  x != 9.9;  x += 0.1) {  // trouble!
      sum += x;
      printf("cnt = %5d\n",  ++cnt);
   }
   printf("sum = %f\n", sum);
   return 0;
}
```

goto 문

- goto 문은 현재 함수 내의 레이블이 붙은 문장으로 무조건 분기함
- 다른 제어 흐름 메커니즘(for, while, do, if, switch)들이 제공하는 모든 유용한 구조를 파괴함
- 따라서 가급적 사용하지 않는 것이 바람직함

레이블 문장

- 레이블 문장

```
bye:  exit(1);
L444: a = b + c;
bug1: bug2: printf("bug found\n");
                    /* multiple labels */
```

- 레이블은 자신의 이름 영역을 가짐
  - 동일한 식별자가 레이블과 변수 모두로 사용될 수 있음

goto 문 예제

```
while (scant("%lf", &x) == 1) {
    if (x < 0.0)
        goto negative_alert;
    printf("%f  %f\n", sqrt(x), sqrt(2 * x));
}
negative_alert:  printf("Negative value
  encountered!\n");
```

break 문

- 루프나 switch 문에서 사용되어 제어를 즉시 루프 다음 문장으로 전달

```
while (1) {
    scanf("%lf", &x);
    if (x < 0.0)
        break;          /* exit loop if x is
  negative */
    printf("%f/n", sqrt(x));
}
/* break jumps to here */
```

continue 문

- 루프 내에서 사용되어 현재 반복을 멈추고 즉시 다음 반복을 하게 함

```
for (i = 0; i < TOTAL;  ++i) {
    c = getchar();
    if (c >= '0' && c <= '9')
      continue;
   .....      /* process other characters */
/* continue transfers control to here to begin
  next iteration */
}
```

- 다음 두 코드는 같은 형태임

```
for (expr1; expr2; expr3)
{
    .....
        continue;
    .....
}
```

```
expr1;
while (expr2) {
    .....
        goto next;
    .....
next :
    expr3;
}
```

switch 문

- switch 문은 if-else 문을 일반화한 다중 조건문
- switch 다음에 오는 괄화 안에 사용되는 제어식은 반드시 정수형
- 제어식의 평가 결과에 따라 제어는 해당되는 case 레이블로 분기
- 예제

```
switch (c) {
case 'a' :
   ++a_cnt;
   break;
case 'b' :
case 'B' :
   ++b_cnt;
   break;
default :
   ++other_cnt;
}
```

- 실행 순서
  1. switch 문의 수식 평가
  2. 단계 1에서 계산된 값과 일치하는 상수 값을 갖는 case 레이블로 분기; 만일 그런 case 레이블이 없다면, default 레이블로 분기; 만일 default 레이블도 없다면, switch 문 종료
  3. break 문을 만나면, switch 문 종료; 또는 switch 몸체의 마지막 문장을 수행하면 switch 문 종료

? : 조건부 연산자

- 삼항 연산자임
- 일반적인 형태

  expr1 ? expr2 : expr3
- 평가 방법

  1. expr1 평가
  2. 참이면, expr2를 평가하고 그 결과가 이 수식의 값이 됨
  3. 거짓이면, expr3을 평가하고 그 결과가 이 수식의 값이 됨
- 이 수식의 값은 expr2와 expr3에 의해 결정됨
  - 만일 이들의 형이 다르다면, 일반적인 변환 규칙이 적용됨
  - 즉, expr2나 expr3 중 어떤 것이 평가되느냐에 의존하지 않음

선언 및 초기화

```
char     a = 'a', b = 'b'; // a has decimal value 97
int      i = 1, j = 2;
double   x = 7.07;
```

수식	동일한 수식	결과값	형
`i == j ? a - 1 : b + 1`	`(i == j) ? (a - 1) : (b + 1)`	99	int
`j % 3 == 0 ? i + 4 : x`	`((j % 3) == 0) ? (i + 4) : x`	7.07	double
`j % 3 ? i + 4 : x`	`(j % 3) ? (i + 4) : x`	5.0	double

▶ x = (y < z) ? y : z;

Declarations and initializations

```
char      a = 'a', b = 'b';      /* a has decimal value 97 */
int       i = 1, j = 2;
double    x = 7.07;
```

Expression	Equivalent expression	Value	Type
i == j ? a -a1 : b + 1	(i == j) ? (a -1) : (b + 1)	99	int
j % 3 == 0 ? i + 4 : x	((j % 3) == 0) ? (i + 4) : x	7.07	double
j % 3 ? i + 4 : x	(j % 3) ? (i + 4) : x	5.0	double

```
x = (y < z) ? y : z;
```

```
if (y < z)
        x =  y ;
else
        x = z;
```

질의 및 응답

-끝-