시스템 프로그래밍 기초

Introduction to System Programming

ICT융합학부 조용우

4. Flow of Control

관계, 등가, 논리 연산자

- 관계 연산자
  - → ~보다 작다: <
  - → ~보다 크다: >
  - → ~보다 작거나 같다: <=
  - → ~보다 크거나 같다: >=
- 등가 연산자
  - → 같다: ==
  - → 같지 않다: !=

관계, 등가, 논리 연산자

- 논리 연산자
  - (단항)부정: !
  - 논리곱: &&
  - 논리합: ||
- 연산 결과
  - false: zero
  - true: nonzero

우선순위, 결합법칙

연산자	결합 법칙
() ++(prefix) --(postfix)	L→R
+(unary) -(unary) ++(prefix) --(prefix)	R→L
* / %	L→R
+ -	L→R
< <= > >=	L→R
== !=	L→R
&&	L→R
||	L→R
?:	R→L
= += -= *= /= etc.	R→L
,	L→R

관계 연산자와 수식

- 관계연산자(이항연산자)
  - < > <= >=
- 예
  - a < 3
  - a > b → (a - b) < 0으로 구현
  - -1.3 >= (2.0 * x + 3.3)
  - a < b < c /* syntactically correct, but confusing */

틀린 예

- `a =< b   /* out of order */`
- `a < = b  /* space not allowed */`
- `a >> b   /* this is a shift expression */`

a - b 의 값과 관계식

Values of relational expressions				
a - b	a < b	a > b	a <= b	a >= b
positive	0	1	0	1
zero	0	0	1	1
negative	1	0	1	0

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

Expression              Equivalent expression          Value
'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
```

해석

- `3 < j < 5`
  - → `(3 < 2) < 5  <==>  0 < 5 (true)`
  - → `3 < j && j < 5`
- `x < x + y`
  - → `(x - (x + y)) < 0.0`
  - → 정밀도 때문에 x와 x + y의 값이 같아서 0

등가 연산자와 수식

- 등가 연산자 ==와 != 는 이항 연산자
- 예
  - `c == 'A'`
  - `k != -2`
  - `x + y == 3 * z - 7`

잘못된 사용 예

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

Values of:		
expr1 - expr2	expr1 == expr2	expr1 != expr2
zero	1	0
non zero	0	1

4.3 Equality Operators and Expressions

```
Declarations and initializations
int       i = 1 , j = 2 , k = 3;

Expression                Equivalent expression          Value
i == j                    j == i                             0
i != j                    j != i                             1
i + j + k == - 2 * - k    ((i + j ) + k) == ((- 2) * (- k))  1
```

Common programming error

```
if (a = 1) /* always true */
    ... /* do something */
```

instead of

```
if (a == 1)
    ... /* do something */
```

논리 부정 연산자 !

- 식이 0 값을 가지면 그것의 부정인 정수값 1 이 생성.
- 식이 0 이 아닌 값을 가지고 있으면 그것의 부정인 정수값 0을 생성.
- 예
  - `!a`
  - `!(x + 7.7)`
  - `!(a < b || c < d)`

잘못 사용된 예

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

Value of:	
`expr`	`!expr`
`zero`	`1`
`nonzero`	`0`

- 논리학 *not* (*not* s) = s
  - → ㄱ(ㄱ S) = S
- **!!5 의 값은 1**
  - → `!(!5)`
  - → `!(0)`
  - → `1`

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

Expression     Equivalent expression     Value
! c            ! c                       0
! (i -j)       ! (i - j)                 1
! i - j        (! i ) - j                -7
! ! (x + y)    ! (! (x + y))             1
! x * ! ! y    (! x) * ( ! ( ! y))       1
```

4.4 Logical Operators and Expressions

- *logical_expression* ::= *logical_negation_expression*
  | *logical_or_expression*
  | *logical_and_expression*
- *logical_or_expression* ::= *expr* || *expr*
- *logical_and_expression* ::= *expr* && *expr*

예

- `a && b`
- `a || b`
- `!(a < b) && c`
- `3 && (-2 * a + 7)`

잘못 사용된 예

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

Values of:			
expr1	expr2	expr1 && expr2	expr1 || expr2
zero	zero	0	0
zero	nonzero	0	1
nonzero	zero	0	1
nonzero	nonzero	1	1

4.4 Logical Operators and Expressions

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

Expression Equivalent expression Value
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
```

단축 평가(short-circuit evaluation)

- 결과가 참인지 거짓인지 판명되면 더 이상 다음 수식을 평가하지 않는다.
- **expr1 && expr2**
  - → expr1이 0면 어차피 0이므로 expr2 평가하지 않음
- **expr1 || expr2**
  - → expr1이 1이면 어차피 1이므로 expr2 평가하지 않음
- 예
  - → `if ((3 < 2) && (4 < 5)) a = 4;`
  - → `if ((3 > 2) || (4 < 5)) a = 4;`

단축평가를 사용한 간단한 예

```
int cnt = 0;
while (++cnt <= 3 && (c = getchar ()) != EOF) {
    ... /* do something */
```

- ++cnt <= 3이 거짓이 되면, 다음 문자는 읽히지 않음
- 결론적으로 세 문자만 처리하게 됨

복합문

- 복합문은 중괄호 {} 로 묶여진 선언문과 실행문.
- *compound_statement* ::= { { *declaration* }$_{0+}$ { *statement* }$_{0+}$ }
- 복합문의 예

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

수식 문장과 공백 문장

- *expr_statement* ::= { *expr* }$_{opt}$;
- 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 */
```

4.7 if와 if-else 문

- `if (`***expr***`)`

  statement

  → 만일 ***expr*** 이 0 이 아니면 (true) ***statement*** 가 실행.

  → 그렇지 않으면 ***statement***는 실행되지 않고 다음 문장으로 넘어감

- ```
  if (grade >= 90)
      printf("Congratulations!\n");
  printf("your grade is %d.\n", grade);
  ```
  ```

올바른 if 문의 예

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

참조

- 잘못된 사용 예

```
if b == a              /* parentheses missing */
      area = a * a;
```

- 더 효율적인 코드

```
if (j < k)
      min = j;
  if (j < k)
      printf(" j is smaller than k\n");
```

```
if (j < k) {
      min = j;
      printf("j is smaller than k\n");
  }
```

if-else 문

- ```
  if (expr)
        statement1
  else
        statement2
  ```
  - → *expr* 이 0 이 아니면 *statement1* 이 실행되고 *statement2* 는 실행 되지 않음
  - → *expr* 이 0 이면 *statement1* 은 건너뛰고 *statement2* 가 실행됨
- ```
  if (x < y)
        min = x;
  else
        min = y;
  ```

올바른 if-else 문의 예

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

참조

- 잘못 사용한 예

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

- else는 가장 가까운 if와 연관됨

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

while 문

- **while (*expr*)**

  statement

  → ***expr***이 **0**이 아니면 ***statement***를 실행한 후, 다시 **while**루프의 시작 부분으로 돌아 감

  → ***expr***이 **0**이면 다음 문장으로 진행함

- **while (i++ < n)**

  factorial *= i;

올바른 while 문의 예

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

참조

- 잘못된 사용 예

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

- 입력 스트림에서 공백 문자 무시

```
while ((c = getchar()) == ' ')
     ;              /* empty statement */
```

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' || 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;
}
```

for 문

- **for (*expr1*; *expr2*; *expr3*)**
  **   *statement***
  - → 우선 ***expr1***을 실행(초기화) 후 ***expr2*** 가 **0**이 아니면, ***statement***를 실행한 후, ***expr3***를 수행함(카운트 증가)
  - → ***expr2***가 **0**이면, 다음 문장으로 진행
- 아래의 while 문과 같다.
  - →
```
expr1;
while (expr2) {
    statement
    expr3;
}
```

예

- 올바른 for 문의 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;
    ```
    ```

1에서 10까지 정수의 합

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

무한루프

```
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;
}
```

4.10 An Example: Boolean Variables

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
4	0	0	0	1	1	1	1	0
5	0	0	1	0	0	1	0	0
6	0	0	1	0	1	1	0	0
7	0	0	1	1	0	1	0	0
8	0	0	1	1	1	1	1	1
9	0	1	0	0	0	0	0	0
10	0	1	0	0	1	1	0	0
11	0	1	0	1	0	0	0	0
12	0	1	0	1	1	1	1	1
13	0	1	1	0	0	1	0	0
14	0	1	1	0	1	1	0	1
15	0	1	1	1	0	1	0	1
16	0	1	1	1	1	1	1	1
17	1	0	0	0	0	1	0	0
18	1	0	0	0	1	1	0	0
19	1	0	0	1	0	1	0	0
20	1	0	0	1	1	1	1	1
21	1	0	1	0	0	1	0	0
22	1	0	1	0	1	1	0	1
23	1	0	1	1	0	1	0	1
24	1	0	1	1	1	1	1	1
25	1	1	0	0	0	1	1	0
26	1	1	0	0	1	1	1	1
27	1	1	0	1	0	1	1	1
28	1	1	0	1	1	1	1	1
29	1	1	1	0	0	1	1	1
30	1	1	1	0	1	1	1	1
31	1	1	1	1	0	1	1	1
32	1	1	1	1	1	1	1	1

콤마 연산자

- ***expr1***, ***expr2***
  - ***expr1***이 먼저 평가되고, 그 다음 ***expr2***가 평가됨
  - 전체 수식의 값과 형은 가장 오른쪽 피연산자를 따름
  - 가장 낮은 우선순위 갖는 이항 연산자
  - L→R 결합 법칙
- 프로그램 상의 대부분의 콤마는 콤마 연산자가 아님
  - 함수의 인자 목록에서 수식을 분리하는데 사용된 콤마
  - 초기화 목록에서 사용된 콤마

(어색한 예) 1에서 n 까지의 정수의 합을 계산

```
for (sum = 0, i = 1; i <= n; ++i)
    sum += i;
for (sum = 0, i = 1; i <= n; sum += i, ++i)  /* correct */
    ;
for (sum = 0, i = 1; i <= n; ++i, sum += i)  /* wrong */
    ;
```

(자연스러운 예) 인덱스와 포인터를 동시에 유지

```
→for (i = 0, p = head; p != NULL; ++i, p = p -> next)
      ...
```

4.11 The Comma Operator

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

Expression              Equivalent expression              Value
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

- **do**

  statement

  while (*expr*);

  - 먼저 ***statement***를 실행한 후, ***expr***이 **0**이 아니면, **do** 문의 시작 부분으로 돌아 감
  - ***expr***이 **0**이면 다음 문장으로 진행
  - **do** 문은 **while** 문의 변형된 형태
  - **while** 문은 루프 상단에서 조건을 검사하는 반면 **do** 문은 루프 하단에서 조건을 검사

(예) 0이 입력되기 전까지 정수를 입력

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

(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");
 } while (error);
```

(주의) 등가 수식 보다는 관계 수식으로 조건 평가

```
/* A test that fails. */
#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;
}
```

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

goto 문

- 현대 프로그래밍 방법론에서 유해한 구조물로 간주!
- 현재 함수 내의 레이블이 붙은 문장으로 무조건 분기함으로써 다른 제어 흐름 메커니즘(**if**, **for**, **while**, **do**, **switch**)들이 제공하는 유용한 구조를 파괴
- When should a **goto** be used?
  - → A simple answer is "not at all."

break 문

- **break**와 **continue**는 정상적인 제어의 흐름을 중단시킴
- **break** 문은 루프의 내부나 **switch** 문으로부터 빠져 나옴
- (예) 제곱근

```
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

- **continue** 문은 **for**, **while**, **do** 루프의 현재 반복 동작을 멈추고 즉시 다음 반복을 하게함
- (예) 숫자를 제외한 모든 문자를 처리

```
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 */
}
```

switch 문

- switch 문은 if-else 문을 일반화한 다중 조건문.
- switch 문의 전형적인 예

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

switch 문

- **switch** 다음에 오는 괄호 안에 사용되는 제어식은 반드시 정수적 형
- 제어식의 평가 결과에 따라 제어는 해당되는 case 레이블로 분기

조건부 연산자

- ***expr1 ? expr2 : expr3***
  - 조건부 연산자 ?: 는 삼항 연산자
  - 3개의 수식을 피연산자로 가짐
  - ***expr1***이 ***0***이 아니면 ***expr2***가 이 조건부 수식의 값이 됨
  - ***expr1***이 ***0***이면 ***expr3***가 이 조건부 수식의 값이 됨
  - 조건부 수식의 형은 ***expr2***와 ***expr3***에 무관하게 둘 다 포함할 수 있는 일반적인 변환 규칙 적용

if-else 문과 조건부 연산자

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

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

4.17 The Conditional Operator

```
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 - 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
```

Homework

- Exercises #1, 11, 13, 19, 33