

Chapter 2. 제어의 흐름

박 종 혁 교수

UCS Lab

(<http://www.parkjonghyuk.net>)

Tel: 970-6702

Email: jhpark1@snut.ac.kr

Learning Objectives

- 부울 식(Boolean Expressions)
 - 구성, 평가 & 우선순위 규칙
- 분기 매커니즘
 - if-else
 - switch
 - 내포된 if-else
- 루프
 - While, do-while, for
 - 내포된 loops

부울 식:

디스플레이 2.1 비교 연산자

- 논리 연산자
 - 논리 곱(Logical AND) : &&
 - 논리 합(Logical OR) : ||

Display 2.1 Comparison Operators

MATH SYMBOL	ENGLISH	C++ NOTATION	C++ SAMPLE	MATH EQUIVALENT
=	Equal to	==	<code>x + 7 == 2*y</code>	$x + 7 = 2y$
≠	Not equal to	!=	<code>ans != 'n'</code>	$ans \neq 'n'$
<	Less than	<	<code>count < m + 3</code>	$count < m + 3$
≤	Less than or equal to	<=	<code>time <= limit</code>	$time \leq limit$
>	Greater than	>	<code>time > limit</code>	$time > limit$
≥	Greater than or equal to	>=	<code>age >= 21</code>	$age \geq 21$

부울 식 평가

- 자료형 bool
 - 참(true) 또는 거짓(false)을 리턴
 - 참과 거짓은 사전 정의된 라이브러리 상수
 - 여러 개의 부울 식을 연결하는 경우에는 괄호 '(')'를 적절하게 사용하는 것이 중요하다!
- 진리표
 - 다음 슬라이드의 디스플레이 2.2

부울 식: 디스플레이 2.2 진리표

Display 2.2 Truth Tables

AND

<i>Exp_1</i>	<i>Exp_2</i>	<i>Exp_1</i> && <i>Exp_2</i>
true	true	true
true	false	false
false	true	false
false	false	false

OR

<i>Exp_1</i>	<i>Exp_2</i>	<i>Exp_1</i> <i>Exp_2</i>
true	true	true
true	false	true
false	true	true
false	false	false

NOT

<i>Exp</i>	! (<i>Exp</i>)
true	false
false	true

디스플레이 2.3

연산자의 우선순위 (1 of 4)

Display 2.3 Precedence of Operators

::	Scope resolution operator
.	Dot operator
->	Member selection
[]	Array indexing
()	Function call
++	Postfix increment operator (placed after the variable)
--	Postfix decrement operator (placed after the variable)
++	Prefix increment operator (placed before the variable)
--	Prefix decrement operator (placed before the variable)
!	Not
-	Unary minus
+	Unary plus
*	Dereference
&	Address of
new	Create (allocate memory)
delete	Destroy (deallocate)
delete[]	Destroy array (deallocate)
sizeof	Size of object
()	Type cast

*Highest precedence
(done first)*

디스플레이 2.3

연산자의 우선순위 (2 of 4)

* / %	Multiply Divide Remainder (modulo)
+ -	Addition Subtraction
<< >>	Insertion operator (console output) Extraction operator (console input)



*Lower precedence
(done later)*

디스플레이 2.3

연산자의 우선순위 (3 of 4)

Display 2.3 Precedence of Operators

All operators in part 2 are of lower precedence than those in part 1.

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal
!=	Not equal
&&	And
	Or

디스플레이 2.3

연산자의 우선순위 (4 of 4)

=	Assignment
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulo and assign
? :	Conditional operator
throw	Throw an exception
,	Comma operator

↓
*Lowest precedence
(done last)*

우선순위 예제

- 논리 연산자 이전의 산술 식
 - $x + 1 > 2 \ || \ x + 1 < -3$ 의 의미:
 - $(x + 1) > 2 \ || \ (x + 1) < -3$
- 단락 회로 평가(Short-circuit evaluation)
 - $(x \geq 0) \ \&\& \ (y > 1)$
 - 증가/감소 연산자에 주의!
 - $(x > 1) \ \&\& \ (y++)$
- 정수(integer) 값은 부울 값으로 평가
 - 모든 0이 아닌 값 \rightarrow true
 - 0 값 \rightarrow false

분기 매커니즘

- if-else 문
 - 조건 식에 근거하여 두 개의 문장 중에서 선택하여 실행
 - 예:
if (hrs > 40)
 grossPay = rate*40 + 1.5*rate*(hrs-40);
else
 grossPay = rate*hrs;

if-else 문장 구문

- 형식 구문:
if (<boolean_expression>
 <yes_statement>
else
 <no_statement>
- 각각의 선택은 단 하나의 문장!
- 각각의 분기에서 다수의 문장을 실행하려면 →
복합문 사용

복합/블록 문

- 각 분기마다 하나의 문장을 가짐
- 복합문을 사용하기 위해서는 반드시 '{}'를 이용한다
 - “블록”문이라 함
- 각각의 블록은 블록문을 가져야 한다
 - 하나의 문장이라도 {}를 사용
 - 가독성 향상

복합문의 사용 예

- 이 예제의 들여쓰기를 주목:

```
if (myScore > yourScore)
{
    cout << "I win!\n";
    wager = wager + 100;
}
else
{
    cout << "I wish these were golf scores.\n";
    wager = 0;
}
```

일반적인 함정

- 연산자 "=" vs. 연산자 "=="
- = : “할당”의 의미를 가짐
- == : “등가”의 의미를 가짐
 - C++에서는 매우 다른 의미이다!
 - 예:
if (x = 12) ← 연산자의 사용을 주목!
 Do_Something
else
 Do_Something_Else

선택 가능한 else

- else 절은 선택 가능
 - 분기문에서 false일 경우, 아무런 실행도 원하지 않는다면 else 부분을 생략해도 된다
 - 예:
if (sales >= minimum)
 salary = salary + bonus;
cout << "Salary = %" << salary;
 - 주목 : else절이 없으므로 조건이 false 일 경우 아무것도 실행되지 않음!
 - 이어서 cout 문이 계속 수행된다

내포된 문장 (Nested Statements)

- if-else 문은 작은 문장을 포함한다
 - 복합문 또는 단순한 문장 (보여지는)
 - 또다른 if-else 문장을 포함하여 어떠한 문장도 포함할 수 있다!
 - 예:

```
if (speed > 55)
    if (speed > 80)
        cout << "You're really speeding!";
    else
        cout << "You're speeding.";
```
 - 적절한 들여쓰기를 주목!

다중 if-else

- 새로운 것이 아니고, 단지 들여쓰기가 다르다
- “과도한” 들여쓰기는 피할 것
 - 구문:

Multiway if-else Statement

SYNTAX

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
    .
    .
    .
else if (Boolean_Expression_n)
    Statement_n
else
    Statement_For_All_Other_Possibilities
```

다중 if-else 예

EXAMPLE

```
if ((temperature < -10) && (day == SUNDAY))  
    cout << "Stay home.";  
else if (temperature < -10) //and day != SUNDAY  
    cout << "Stay home, but call work.";  
else if (temperature <= 0) //and temperature >= -10  
    cout << "Dress warm.";  
else //temperature > 0  
    cout << "Work hard and play hard.";
```

The Boolean expressions are checked in order until the first true Boolean expression is encountered, and then the corresponding statement is executed. If none of the Boolean expressions is true, then the *Statement_For_All_Other_Possibilities* is executed.

switch 문

- 다중 분기의 제어를 위한 새로운 문장
- 부울 데이터 형을 리턴하는 제어 문장을 사용 (true 또는 false)
- 구문:
 - 다음 슬라이드

switch 문의 구문

switch Statement

SYNTAX

```
switch (Controlling_Expression)
{
    case Constant_1:
        Statement_Sequence_1
        break;
    case Constant_2:
        Statement_Sequence_2
        break;
        .
        .
        .
    case Constant_n:
        Statement_Sequence_n
        break;
    default:
        Default_Statement_Sequence
}
```

*You need not place a **break** statement in each case. If you omit a **break**, that case continues until a **break** (or the end of the **switch** statement) is reached.*

switch 문의 활용 예

EXAMPLE

```
int vehicleClass;
double toll;
cout << "Enter vehicle class: ";
cin >> vehicleClass;

switch (vehicleClass)
{
    case 1:
        cout << "Passenger car.";
        toll = 0.50;
        break;
    case 2:
        cout << "Bus.";
        toll = 1.50;
        break;
    case 3:
        cout << "Truck.";
        toll = 2.00;
        break;
    default:
        cout << "Unknown vehicle class!";
}
```

*If you forget this break,
then passenger cars will
pay \$1.50.*

switch 문: 다중 case 레이블

- break를 만날 때까지 하위 문장을 연속적으로 실행
 - switch 는 “선택 점”을 제공
 - 예:

```
case "A":  
case "a":  
    cout << "Excellent: you got an "A"!\n";  
    break;  
case "B":  
case "b":  
    cout << "Good: you got a "B"!\n";  
    break;
```
 - 다중 레이블은 같은 “선택”을 제공함을 주목!

switch 함정/팁

- break의 누락 :
 - 컴파일 에러는 발생하지 않음
 - break를 만날 때까지 다른 case의 문장을 연속적으로 실행
- 가장 많은 사용처 : 메뉴
 - 명확한 “전체적 관점”을 제공
 - 메뉴의 구조를 효과적으로 보여줌
 - 각각의 분기는 하나의 메뉴 선택에 대응된다

switch 메뉴 예

- 메뉴를 위한 완전한 switch 문:

```
switch (response)
{
    case "1":
        // Execute menu option 1
        break;
    case "2":
        // Execute menu option 2
        break;
    case "3":
        // Execute menu option 3
        break;
    default:
        cout << "Please enter valid response.";
}
```

나열형 (enumeration type)

- 정수형 상수 리스트 정의
 - `enum MonthLength {JAN=31, FEB=28, MAR=31, APR=30, MAY=31, JUN=30, JUL=31, AUG=31, SEP=30, OCT=31, NOV=30, DEC=31};`
 - `enum Direction {NORTH, SOUTH, EAST, WEST};`
 - `enum Direction {NORTH=0, SOUTH=1, EAST=2, WEST=3};`
- 정수로서 표현되지만 정수형과는 다른 유형이므로 산술계산은 하지 않는 것이 좋다.

조건 연산자

- “삼항 연산자”라고도 한다
 - 식에 조건이 삽입된 형태
 - 본질적으로 “축약된 if-else” 연산자
 - 예:
if (n1 > n2)
 max = n1;
else
 max = n2;
 - 다음과 같이 작성 가능:
max = (n1 > n2) ? n1 : n2;
 - 삼항 연산자의 형식 : "?" 와 ":" 을 사용

순환문(Loops)

- C++의 3가지 형태의 순환문
 - while
 - 가장 유연함
 - “제약”이 없다
 - do-while
 - 가장 유연하지 않음
 - 항상 최소한 한번 루프 바디(loop body)가 실행된다
 - for
 - 자연적인 “카운팅(counting)” 루프 : n번 반복되는 루프
 - 루프 바디(loop body) : 순환문에서 반복 실행되는 구문

while 루프 구문

Syntax for while and do-while Statements

A while STATEMENT WITH A SINGLE STATEMENT BODY

```
while (Boolean_Expression)  
    Statement
```

A while STATEMENT WITH A MULTISTatement BODY

```
while (Boolean_Expression)  
{  
    Statement_1  
    Statement_2  
    .  
    .  
    .  
    Statement_Last  
}
```

while 루프 예

- 다음을 고려해보면:

```
count = 0;           // 초기화
while (count < 3)     // 루프 조건
{
    cout << "Hi ";    // 루프 바디
    count++;          // 갱신 구문
}
```

- 루프 바디는 몇 번 실행되는가?

do-while 루프 구문

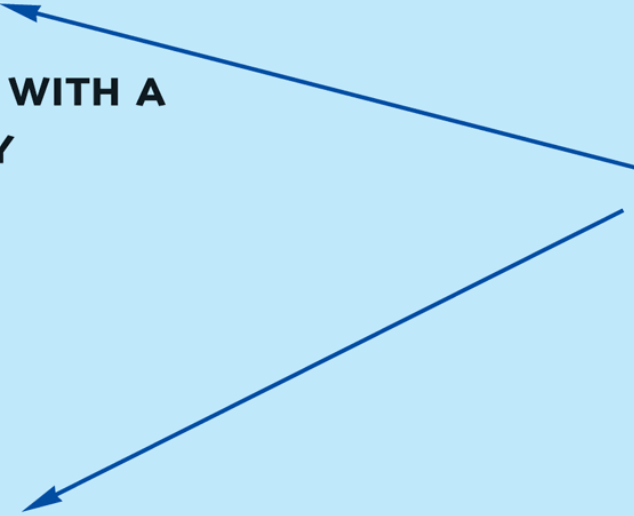
A do-while STATEMENT WITH A SINGLE-STATEMENT BODY

```
do  
    Statement  
while (Boolean_Expression);
```

A do-while STATEMENT WITH A MULTISTatement BODY

```
do  
{  
    Statement_1  
    Statement_2  
    .  
    .  
    .  
    Statement_Last  
} while (Boolean_Expression);
```

*Do not forget
the final
semicolon.*



do-while 루프 예

- ```
count = 0; // 초기화
do
{
 cout << "Hi "; // 루프 바디
 count++; // 갱신 구문
} while (count < 3); // 루프 조건
```

  - 루프 바디는 몇 번 실행되는가?
  - do-while 루프의 바디는 최소한 한번 실행된다!



# while vs. do-while

- 매우 유사하다. 그러나...
  - 한가지 중요한 차이점
    - “언제” 부울식이 체크되는지가 초점
    - while: 루프 바디의 실행 전에 체크
    - do-while: 루프 바디가 실행된 후 체크
- 이 차이점 외에는 근본적으로 같다!
- while 의 “유연성” 때문에 더욱 일반적으로 사용된다

# 컴마 연산자

- 식의 리스트를 평가하고, 마지막 식의 값을 리턴한다
- for 루프에 가장 많이 사용
- 예 :  
first = (first = 2, second = first + 1);
  - first 에 값 2가 할당
  - second 에 값 3이 할당
- 어떠한 순서로 식이 평가되는지에 대한 보장은 없다 : 컴파일러에 따라 다름
- 3개 이상의 식의 연결도 가능 : 괄호를 이용하여 순서를 명시

# for 루프 구문

```
for (Init_Action; Bool_Exp; Update_Action)
 Body_Statement
```

- if-else와 같이, Body\_Statement는 블록문 사용이 가능
  - 좀 더 일반적이다

# for 루프 예

- ```
for (count=0;count<3;count++)  
{  
    cout << "Hi ";    // 루프 바디  
}
```
- 루프 바디는 몇 번 실행되었는가?
- 초기화, 루프 조건 및 갱신 부분 모두 “for 루프 구조 내”에 위치한다!
- 자연적인 “카운팅(counting)” 루프 : n번 반복되는 루프

루프 이슈

- 루프의 조건 식은 “어떠한” 부울 식이라도 가능
- 예:

```
while (count<3 && done!=0)
{
    // Do something
}
for (index=0;index<10 && entry!=-99)
{
    // Do something
}
```

루프 함정 : ';'의 잘못된 위치

- ';' (세미콜론)'의 잘못된 위치
 - 예:

```
while (response != 0) ;←  
{  
    cout << "Enter val: ";  
    cin >> response;  
}
```
 - While문의 조건 뒤에 ';'이 위치!
- 결과 : 무한 루프(INFINITE LOOP)!

루프 함정 : 무한 루프

- 수행 중에 루프의 조건이 거짓이어야 루프가 종료됨
 - 그렇지 않으면 무한 루프!
 - 예:

```
while (1)
{
    cout << "Hello ";
}
```
 - 완벽하게 C++의 문법을 만족하는 루프 → 항상 무한!
- 무한 루프는 올바르게 사용할 수 있음
 - 임베디드 시스템

break 와 continue 문

- 제어의 흐름
 - 루프가 제공하는 완벽한 입/출력 제어의 흐름에 삽입된다
 - 몇몇 드문 예에서, 자연스러운 흐름을 바꿀 수 있음
- break;
 - 즉시 루프를 빠져나옴.
- continue;
 - 루프 바디의 나머지 부분을 생략
- 이들 문장은 자연스런 흐름을 파괴
 - 반드시 필요한 경우에만 사용!

내포된 루프(Nested Loops) : 중첩된 순환문

- 어떠한 유효한 C++의 문장도 루프 바디 안에 삽입 가능
- 추가적인 루프 문장 또한 가능!
 - “내포된 루프(nested loops)” : 중첩된 순환문
- 세심한 들여쓰기가 필요:
for (outer=0; outer<5; outer++)
 for (inner=7; inner>2; inner--)
 cout << outer << inner;
 - 각각의 바디가 하나의 문장인 경우에는 {}가 없음
 - 항상 {}를 사용하는 것이 좋은 스타일 : 가독성 향상

요약 1

- 부울 식
 - 산술식과 유사 → 결과는 true 또는 false
- C++ 분기문
 - if-else, switch
 - switch 문은 메뉴에 적합
- C++ 순환문
 - while
 - do-while
 - for

요약 2

- do-while 루프
 - 항상 루프 바디는 최소한 한번은 실행된다
- for 루프
 - 자연적인 “카운팅(counting)” 루프 : n번 반복되는 루프
- 루프에서 조기에 빠져나갈 수 있다
 - break문
 - continue 문
 - 자연스러운 흐름을 위해 엄격하게 사용

Q&A