Control Structure
(조건문, 반복문)

goto문 쓰면 컨트롤이 엉망이 됨
goto문 있으면 순서가 엉망이 되어 가독성이 떨어짐

국민대학교 컴퓨터공학부
강 승 식

제어구조(control structure)

- Control structure는 multiple entry를 허용하는가?
  - Single entry single exit
  - Single entry multiple exit 대부분이 이거
  - Multiple entry multiple exit

c언어 함수의 중간으로는 goto 안됨
한 함수 내에선 multiple entry 가능

if문 중간에 goto해서 들어갈수가있냐
for문 중간에 goto해서 들어갈수있냐

c언어는 goto가능
특이한언어
Multiple

Selection Statements(조건문)

- Two-way selectors
  - if-then-else

- Multiple-way selectors
  - 파이썬: if-elsif-elsif-…-else if-elif-elif-else 오타임
  - C 언어: switch(or case) statement

Two-Way Selection Statements

- General form:

  if control_expression
  then clause
  else clause

참이면
거짓이면

차이점이라면
C언어는 if 다음에 괄호 있고
파이썬은 if 다음에 괄호 없음 또한 콜론있음

if 다음에 then이 있으면 괄호 필요 x

- Design Issues:
  1. Form and type of the control expression(제어식)
     Form -- ( ) 안에 기술 or ':' 으로 끝남(Python)
     제어식 -- 정수 or 불린 타입
  2. *then* and *else* clauses 기술 방법
     { } 사이에 기술 or 탭 문자로 시작(python)
  3. Meaning of nested selectors: else 모호성

C언어만 {}지 보통은
Begin end임 – 실용적인거보다는
외형적인거 중시(시간이 좀 많이드는 단점이
있어서 C언어가 보편화가 됨)

Example

- Java

```
if (sum == 0)
   if (count == 0)
       result = 0;
else result = 1;
```

실용성 위주

- Ruby

```
if sum == 0 then
   if count == 0 then
     result = 0
   else
     result = 1
   end
end
```

then 폼 위주

그자체가 실행하는게 아닌것은
일관성 있게 콜론을 씀

- Python

```
if sum == 0 :
   if count == 0 :
      result = 0
   else :
      result = 1
```

실제 실행하는문자는
세미콜론이 없음

Selector Expressions

- In ML, F#, and LISP, the selector is an expression

- F#

```
let y =
    if x > 0 then x
    else 2 * x
```

Multiple-Way Selection Statements

- Design Issues:

수식의 결과가 분명한 타입으로 나오는경우???

1. What is the form and type of the control expression?
2. How are the selectable segments specified?
3. Is execution flow through the structure restricted to include just a single selectable segment?
4. How are case values specified?
5. What is done about unrepresented expression values?

case 끝나면 끝나냐? c언어는 break; 있어야함

case 1,3,5 가능하냐? C언어는 한개밖에 안댐
다른언어는 가능

Switch 문: C, C++, and Java

multiple way

```
switch (expression) {
        case const_expr_1: stmt_1;
        …
        case const_expr_n: stmt_n;
        [default: stmt_n+1]
}
```

default문이 없는 언어도 있음
c언어는 있음

case사이에 break 안쓰는 경우
case 1,3,7일때 실행하고싶을때
case 1:
3:
7:
실행문
break;
다른언어는 case 1,3,7;

- Design choices for C's switch statement
  1. Control expression can be only an integer type
  2. Selectable segments
     statement sequences, blocks, or compound statements
  3. Any number of segments can be executed: no implicit branch at the end
  4. Default clause is for unrepresented values

C언어는 expression이 무조건 정수여야함

c언어는 char형도 실제론 정수다?

다른언어는 보통 break가 없음

- C# C언어처럼 break없다고 내려가지않음 딱 한블럭만 사용함
  - Disallows the implicit execution of more than one segment
  - Each selectable segment must end with an unconditional branch (**`goto`** or **`break`**)
  - The case constants can be strings 스트링도 됨
- Ada Ada는 case 수식 when when when

```
case expression is 괄호 필요 x
    when choice list => stmt_sequence;
    …
    when choice list => stmt_sequence;
    when others => stmt_sequence;]
end case;
```

- Ada design choices:
  1. Expression can be any ordinal type
  2. Segments can be single or compound
  3. Only one segment can be executed per execution of the construct
  4. Unrepresented values are not allowed

- Constant List Forms:
  1. A list of constants 1,2,3 허용
  2. Can include:
     - Subranges
     - Boolean OR operators (|)

Ruby: two forms of case statements

1. One form uses when conditions

```
leap = case
       when year % 400 == 0 then true
       when year % 100 == 0 then false
       else year % 4 == 0
       end
```

2. The other uses a case value and when values

```
case in_val
when -1 then neg_count++
when 0 then zero_count++
when 1 then pos_count++
else puts "Error – in_val is out of range"
end
```

- Python

```
if count < 10 :
  bag1 = True
elif count < 100 :
  bag2 = True
elif count < 1000 :
  bag3 = True
```

- Ruby

```
case
  when count < 10 then bag1 = true
  when count < 100 then bag2 = true
  when count < 1000 then bag3 = true
end
```

- Scheme

```
(COND
   (predicate1 expression1)
    …
   (predicaten expressionn)
   [(ELSE expressionn+1)]
)
```

Iterative Statements

- The repeated execution is accomplished either by iteration or recursion

- General design issues
  1. How is iteration controlled?
  2. Where is the control mechanism in the loop?

Counter-Controlled Loops

- loop variable, initial, terminal, and stepsize values

- Design Issues:
  1. What are the type and scope of the loop variable?
  2. Should it be legal for the loop variable or loop parameters to be changed in the loop body, and if so, does the change affect loop control?
  3. Should the loop parameters be evaluated only once, or once for every iteration?

- Ada

  for var in [reverse] discrete_range loop

  ...

  end loop

- Design choices:
  - Type of the loop variable is that of the discrete range

    (A discrete range is a sub-range of an integer or enumeration type).
  - Loop variable does not exist outside the loop 바깥으로 벗어나면 존재할수 없음
  - The loop variable cannot be changed in the loop, but the discrete range can; it does not affect loop control 루프안에서 루프변수가 바뀔 수 없음
  - The discrete range is evaluated just once 평가는 반복 준비할때 딱 한번만 한다.
  - Cannot branch into the loop body

c언어의 모든 for문은 while로 바꿀수있음
expr1 while 앞에, 가운데꺼 냅주고, expr3 마지막으로옮기면
똑같아진다. but for문은 가운데꺼아무것도없으면 무한루프
but while()안에는 1이 있어야 무한루프 돔 for문은 보통 카운트할때 씀
while은 카운트할일은 없지만 수식이 참인경우에만 반복실행할때 씀

- C-based languages

for ([expr_1] ; [expr_2] ; [expr_3]) statement

  - The expressions can be whole statements, or even statement sequences, with the statements separated by commas
  - The value of a multiple-statement expression is the value of the last statement in the expression
  - If the second expression is absent, it is an infinite loop

- Design choices:
  - There is no explicit loop variable 분명하게 보이는 루프변수가 없다
  - Everything can be changed in the loop
  - The first expression is evaluated once, but the other two are evaluated with each iteration
  - It is legal to branch into the body of a for loop in C

- C++ differs from C in two ways:
  - The control expression can also be Boolean
  - The initial expression can include variable definitions (scope is from the definition to the end of the loop body)

- Java and C#
  - Differs from C++ in that the control expression must be Boolean 제어식이 불린이다

- Python

  for loop_variable in object:
  - loop body
  [else:
  - else clause]

특이한점 else가 있음
for문이 정상적으로 끝났을때
else가 실행됨
비정상적으로 끝나면 else 실행안됨
그래서 for문이 정상적끝났는지 아닌지 파이썬은 파악할 수 있다.
만약 밑에 2,4,6 실행해야되는데 break 만나면 비정상적 실행이라 할수있음

- The object is often a range
  - a list of values in brackets ([2, 4, 6])
  - a call to the range function -- range(5) returns 0, 1, 2, 3, 4
- The loop variable takes on the values specified in the given range, one for each iteration
- The else clause, which is optional, is executed if the loop terminates normally

Logically-Controlled Loops

- Repetition control is based on a Boolean expression

- Design issues:
  - Pretest or posttest?
  - Should the logically controlled loop be a special case of the counting loop statement or a separate statement?

프리테스트 = 반복하기전에 테스트// posttest 반복후 테스트
식당예로들면 밥먹기전에 돈내냐 밥벅고나서 돈내냐

- C and C++ have both pretest and posttest forms, in which the control expression can be arithmetic:

```
while (control_expr)        do
    loop body                   loop body
                            while (control_expr)
```

pretest

posttest

c언어는 불린타입없음 0,0이아닌 경우로 판별

  - In both C and C++ it is legal to branch into the body

c,c++은 goto (낙하산) 다 허용
java는 control expression이 불린타입이여야만함

- Java is like C and C++, except the control expression must be Boolean and the body can only be entered at the beginning -- Java has no goto

User-Located Loop Control Mechanisms

- loop control (other than top or bottom of the loop)
  - Simple design for single loops (e.g., break)
- Design issues for nested loops
  - Should the conditional be part of the exit?
  - Should control be transferable out of more than one loop?
- C , C++, Python, Ruby, and C# have unconditional unlabeled exits (break)
- Java and Perl have unconditional labeled exits (break in Java, last in Perl)
- C, C++, and Python have an unlabeled control statement, continue
- Java and Perl have labeled versions of continue

break 만나면 그 그문만 딱 벗어남
컨티뉴는 계속반복

자바는 레이블없어서 labeled break or labeled continue가 필요하다 자바는 goto없음

Unconditional Branching

- Some languages do not support goto statement (e.g., Java)

- C#
  - goto statement can be used in switch statements

- Loop exit statements are restricted and somewhat camouflaged goto's

Guarded Commands, by Dijkstra

goto없는 세상에서 살자!! 주장함
os시간때 배운 식탁 철학자 젓가락
그거 생각한사람임

- Selection Guarded Command

시작 If <Boolean expr> -> <statement>
[ ] <Boolean expr> -> <statement>
... 조건식 만족하면 실행할것
[ ] <Boolean expr> -> <statement>
fi

끝 표시 if뒤집어놓은거

모든조건문은 이걸로다 구현 가능함
세세한건 니가 알아서해라~~

- Loop Guarded Command

expr은 빠진것임 원래 넣어야함 오타

do <Boolean> -> <statement>
[ ] <Boolean> -> <statement>
...
[ ] <Boolean> -> <statement>
od

BNF EBNF 똑같음 if 랑 do랑
시멘틱스만 다름 루프는 참일때까지 반복이므로