02_조건문

- 조건문
  - 조건에 따라 둘 또는 그 이상의 실행 경로 중에서 하나를 선택할 수 있는 수단을 제공하는 문장
  - 조건이 참이냐 거짓이냐에 따라 선택하는 if 문
  - 조건에 따라 여러 경로 중 하나를 선택하는 case(or switch) 문
- if 문
  - FORTRAN에서 처음 도입

```
① IF( 식 ) L1, L2
② IF( 식 ) L1, L2, L3
```

① 식이 참이면 레이블 L1로 분기하고, 거짓이면 레이블 L2로 분기
② 식의 값이 음수이면 레이블 L1로, 0이면 레이블 L2로, 그리고 양수이면 레이블 L3으로 분기

02_조건문

- 새롭게 도입된 IF 문

```
IF( 식 ) 문장
```

  - 식이 참이면 문장을 실행하고 거짓이면 문장을 실행하지 않음
- 예

```
IF(I .GT. 10) K =20
```

  - 참인 경우에 한 문장만 실행 가능
  - 조건식이 참인 경우 실행될 문장이 여러 개인 경우에는 GoTo 문을 활용해야 하는 단점이 있음

```
10  IF(식) GO TO 20
    문장1
    문장2
    GO TO 10
20  문장3
```

02_조건문

- FORTRAN 77
  - Go To 문 사용 문제를 보완 ➔ 블록 if 문 추가

```
IF(식) THEN
  문장들
ELSE
  문장들
ENDIF
```

  - THEN 다음에 나오는 문장들은 then 절
  - ELSE 절 다음에 나오는 문장들은 else 절
  - 식이 참이면 then 절을 실행하고 거짓이면 else 절을 실행

- 예

```
IF(L .GT. M) THEN
  ISUM = ISUM + L
  LCNT = LCNT + 1
ELSE
  ISUM = ISUM + M
  MCNT = MCNT + 1
ENDIF
```

02_조건문

- C 예

```
if (식)
   문장 1;
else
   문장 2;
```

  - EBNF <if> → if(<expression>) <statement> [else <statement>]
  - 중괄호 사용
    - 식이 참이거나 거짓인 경우에 실행될 문장이 여러 개라면 중괄호로 묶은 복합문을 사용

```
if(식) {
   문장1;
   문장2;
   ⋮
} else {
   문장n;
   문장n+1;
   ⋮
}
```

02_조건문

- C의 dangling else 문제
  - else가 어떤 if와 연결되는지 모호함

```
if ( 식 1 )
if ( 식 2 )
   문장 1;
else
   문장 2;
```

➔

```
if ( 식 1 ){
   if ( 식 2 )
      문장 1;
}else
   문장 2;
```

  - Pascal과 C
    - 'else는 연결된 else가 없는 가장 가까운 if와 결합한다'는 규칙을 적용
    - 첫 번째 if와 연결을 시키려면 중괄호로 두 번째 if 문을 묶음

02_조건문

- if 절에는 의미있는 문장을 배치
  - 조건이 거짓인 경우에 문장을 처리하는 경우
    - if 조건을 부정, 처리해야 할 문장을 if 절로 이동, else 절을 지우기

```
if(strcmp (name, "korea"))
   ;
else {
   /* 처리해야 할 문장*/
}
```

➔

```
if (!strcmp (name, "korea"))
  /* 처리해야 할 문장 */
```

- else 뒤에 if 문 사용 가능

① C 언어

```
if(식 1)
   문장 1;
else if (식 2)
   문장 2;
else
   문장 3;
```

02_조건문

- Ada의 if 문
  - FORTRAN과 유사하게 if 문 끝에 endif 위치
  - then 절과 else 절에는 둘 이상의 문장이 올 수 있음

```
if 식 then
    문장들
else
    문장들
end if;
```

  - Ada의 elseif 문

```
if 식1 then 문장들
else if 식 2 then 문장들
    else 문장들
    end if;
end if;
```

02_조건문

- case 문과 switch 문
  - 조건에 따라 여러 경로 중 하나를 선택
  - ALGOL W에서 처음 도입

```
case 식 of
begin
    문장 1; 문장 2; ...; 문장 n
end
```

- 식의 값이 1이면 문장1
- 2면 문장 2, ..., n이면 문장 n이 선택되어 실행

  - C와 Java

```
switch(식) {
   case 상수1:
      문장들1
   case 상수2:
      문장들2
      :
   default:
      문장들n
}
```

- 식의 값이 상수1이면 문장들1, 문장들2, ..., 문장들n이 실행
- 상수 2이면 문장들2, ..., 문장들n이 실행
- 일치하는 상수가 없을 경우에는 문장들 n이 실행
- default가 없으면 아무 일도 하지 않고 switch 구조를 빠져나옴

02_조건문

- break
  - break를 만나면 switch 구조를 벗어남

```
switch(식){
  case 상수1:
     문장들1
     break;
  case 상수2:
     문장들2
     break;
    :
  default:
     문장들n
}
```

- 식의 값이 상수1이면 문장들1 실행
- 상수 2이면 문장들2 실행
- 일치하는 상수가 없으면 문장들 n 실행

02_조건문

- Ada의 case 문

```
case 식 is
  when 선택리스트1 => 문장들1
  when 선택리스트2 => 문장들2
     :
  when others => 문장들n
end case;
```

- when others는 생략 가능
- 선택리스트들이 식에서 나타날 수 있는 결과를 모두 포함하고 있어야 함
- 만약 모든 결과를 선택리스트에 나타낼 수 없으면 others를 이용
- 상수는 물론 3|5|7과 같은 상수들의 나열, 90..100과 같은 범위도 가능

  - 일치하는 선택리스트가 없을 때 아무 작업도 실행하고 싶지 않다면 null을 이용

```
case 식 is
     :
  when others => null
end case;
```

02_조건문

- case 문을 사용하는 Ada 예제

```
01  with TEXT_IO;
02  use TEXT_IO;
03  procedure caseGrade is
04        package INT_IO is new TEXT_IO.INTEGER_IO (integer);
05        use INT_IO;
06        subtype scoreRange is integer range 0..100;
07        score: scoreRange;
08        grade: character;
09  begin
10        put("score: ");
11        get(score);
12        case score is
13              when 90..100 => grade := 'A';
14              when 80..89 => grade := 'B';
15              when 70..79 => grade := 'C';
16              when 60..69 => grade := 'D';
17              when 0..59 => grade := 'E';
18        end case;
19        put(grade);
20  end caseGrade;
```

03_반복문

- 반복문
  - 특정 부분을 반복해서 실행되게 하는 문장 : while 문, for 문
- FORTRAN의 DO 문
  - 변수가 초기값을 갖고 한 번씩 반복할 때마다 증가값만큼 증가되면서 종료값보다 작거나 같은 동안 '문장들'을 실행
  - 증가값은 생략 가능, 생략하면 반복할 때마다 변수 값은 1씩 증가

```
DO 레이블 변수=초기값, 종료값 [, 증가값]
    문장들
레이블 CONTINUE
```

- 예1

```
DO 10 I=1, 5
    문장들
10 CONTINUE
```

예 2

```
DO 10 I=1, 5, 2
    문장들
10 CONTINUE
```

03_반복문

- while 문
  - 식이 참인 동안 문장을 반복해서 실행
  - C/C++/Java의 while 문

```
while ( 식 )
  문장들 ;
```

  - EBNF

```
<while> → while (<expression>) <statement>
```

  - Ada의 while 문

```
while ( 식 ) {
   문장 1;
   문장 2;
   :
}
```

03_반복문

- while 문을 사용하는 Ada 예제

```
with TEXT_IO;
use TEXT_IO;
procedure sum is
     package INT_IO is new TEXT_IO.INTEGER_IO (integer);
     use INT_IO;
     index, result: integer;
begin
     index := 1;
     result := 0;
     while index <= 10 loop
          result := result + index;
          index := index + 1;
     end loop;
     put(result);
end sum;
```

02_조건문

- do 문

```
do
   문장 ;
while (식);
```

  - while로 나타내기

```
문장;
while (식);
   문장;
```

  - do 문에서는 반복 문장이 단일 문장일지라도 중괄호로 묶는 것이 바람직

```
do{
   문장들 ;
} while (식);
```

03_반복문

- 반복 내의 임의의 지점에서 종료
  - C/C++/Java는 break

```
while(1) {
   ⋮
  if(…) break;
   ⋮
}
```

  - Ada는 exit

```
loop
   ⋮
  exit when …;
   ⋮
end loop;
```

03_반복문

- for 문

```
for (식 1; 식 2; 식 3)
  문장;
```

- 식1 : 초기화
- 식2 : 종료 조건을 판단하는 식
- 식3 : '문장' 실행 후에 평가, 반복 변수의 값을 변환할 때 사용

- EBNF

```
<for> → for ([<expression>];[<expression>];[<expression>]) <statement>
```

- 과정

① 식1 평가
② 식2 평가 : 참이면 '문장'을 실행하고 거짓이면 for 구조를 종료
③ 식2가 참일 때 '문장'을 실행
④ 식3을 평가하고 다시 식2를 평가

➔ 이러한 동작은 식2가 참인 동안 반복

03_반복문

- 항상 참인 경우

```
for ( ; ; )
  문장;
```

- while 문으로 표현

```
식1;
while(식2){
  문장;
  식3;
}
```

- for 문의 사용
  - 반복 변수를 사용해서 임의의 횟수만큼 반복할 때 주로 사용

```
for(i=0; i<size; i++)
  sum = sum + data[i];
```

  - for 문에서의 변수 선언(C, Java, C++)

```
for(int i=0; i<size; i++)
  sum = sum + data[i];
```

03_반복문

- Ada의 for

```
for 변수 in [reverse] 범위 loop
   문장들
end loop;
```

  - 변수 : 범위의 하한값~상한 값이 될 때까지 반복
  - 반복할 때마다 변수의 값은 1씩 증가
  - 범위 : 1..10과 같이 정수 또는 열거 타입의 부분 범위
  - reverse를 사용하면 범위의 값이 역순으로 변수에 배정

- 예 1

```
01  for i in 0..size-1 loop
02     sum := sum + data(i);
03  end loop;
```

- Ada for 문의 특징
  - 반복 변수가 for 구조 내에서만 사용되는 지역 변수

03_반복문

- Ada for 문의 특징
  - 반복 변수가 for 구조 내에서만 사용되는 지역 변수

```
01  i: float := 3.1;
02  for i in 0..5 loop
03     sum := sum + i;
04  end loop;
05  i := 7.9;
```

- Ada 반복 변수
  - 반복 구조 내부에서 값을 임의로 배정할 수 없음

```
01  for i in 0..5 loop
02     sum := sum + i;
03     i := 3;
04  end loop;
```

04_무조건 분기문

- 프로그램의 실행 순서를 특정 위치로 바꾸는 문장
  - goto 문 : FORTRAN과 같은 언어에서 중요한 역할
  - 예
    - FORTRAN

```
10  IF(K(I) .EQ. 0) GO TO 20
    I = I+1
    GO TO 10
20    :
```

  - goto 문 사용 시 주의할 점
    - goto 문을 무분별하게 사용하면 프로그램을 판독하기가 힘들어짐
    - 프로그램의 신뢰성이 상당히 떨어지므로 특별한 경우에만 사용 권장

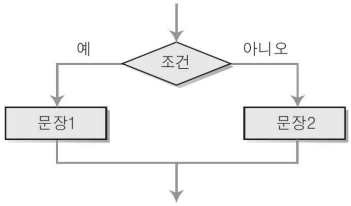
05_구조적 프로그래밍

- 구조적 프로그래밍
  - 다익스트라가 발표한 “Notes on Structured Programming” 논문에서 유래
  - 오직 하나의 입구와 출구만이 있는 제어 구조를 사용해야 한다는 프로그래밍 설계 기법
  - 프로그램을 복잡하게 하는 goto 문은 이용하지 않고 구조화된 순차, 선택, 반복 제어 구조만을 이용하여 프로그램을 설계
  - 구조적 프로그래밍의 핵심을 이루고 있는 순차, 선택, 반복 구조

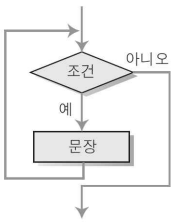
05_구조적 프로그래밍

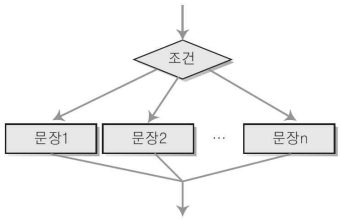
- 순차 구조

선택 구조



- 반복 구조





요약

1. 식은 계산을 표현하는 기본적인 수단으로, 연산자, 피연산자, 괄호, 함수 호출 등으로 구성된다.
2. 연산자는 한 개의 피연산자를 갖는 단항 연산자, 두 개의 피연산자를 갖는 이항 연산자, 세 개의 피연산자를 갖는 삼항 연산자로 구분할 수 있다.
3. 연산자 평가 순서, 피연산자 평가 순서에 따라 결과가 달라질 수 있다.
4. 식의 단락회로 평가란 모든 피연산자와 연산자를 평가하지 않고서도 식의 결과가 결정되는 것을 의미한다.
5. 중복 연산자란 하나의 기호가 두 가지 이상의 목적으로 사용되는 연산자를 의미한다.
6. 조건문은 조건에 따라 둘 또는 그 이상의 실행 경로 중에서 하나를 선택할 수 있는 수단을 제공하는 문장이다.
7. 반복문은 특정 부분을 반복해서 실행되게 하는 문장이다.
8. 무조건 분기문은 프로그램의 실행 순서를 특정 위치로 바꾸는 문장이다.
9. 구조적 프로그래밍은 오직 하나의 입구와 출구만이 있는 제어 구조를 사용해야 한다는 프로그래밍 설계 기법으로, 프로그램을 복잡하게 하는 goto 문은 이용하지 않고 구조화된 순차, 선택, 반복 제어 구조만 이용한다.