Control Structures

2023. Spring

국민대학교 소프트웨어학부
최 준수

문장(statement)

- 문장(혹은 문)
  - 컴퓨터로 하여금 어떤 행동을 취하게끔 하는 가장 작은 단위의 명령
  - 프로그램은 여러 개의 문장으로 구성됨
    - 문장은 쓰여진 순서대로 실행됨
    - 문장의 종류에 따라 실행 순서는 각각 다르게 제어됨
    - block을 제외한 모든 문장은 세미콜론 ; 으로 문장의 끝을 표시해야 함
  - 문장의 종류

복합문	블록(block)
수식문	수식(expression) 문장 빈(empty) 문장
선택문	if 문장 if-else 문장 switch 문장
반복문	for 문장 while 문장 do-while 문장
점프문	break 문장 continue 문장 return 문장
선언문	변수, 함수 선언 문장

블록(block)

- 블록(block)
  - 여러 개의 문장이나 변수 선언문이 중괄호 {}로 묶여 있는 문장
    - 블록은 한 개의 문장 역할
    - 중괄호 안에 아무런 문장이 없는 빈 블록(empty block)도 가능함
    - 블록 내의 여러 문장은 가장 위 문장부터 마지막 문장까지 순서대로 실행됨
    - 블록 내의 선언문은 어느 곳에 위치해도 됨

```
if (a > b)
{
  int t;     // 선언문

  t = a;     // 대입문
  a = b;     // 대입문
  b = t;     // 대입문
}
```

블록

블록
(함수 몸체)

```
int main()
{
  int a = 1;
  cout << "변수 a의 값은 << a << " 입니다." << endl;

  int b = 2;
  cout << "변수 b의 값은 << b << " 입니다." << endl;
}
```

수식 문장(expression statement)

- 수식 문장(expression statement)의 종류
  - 수식으로 표현된 문장
    - 수식과 문장의 끝을 표시하는 세미콜론으로 구성된 문장
  - C++ 프로그램의 대부분의 문장은 수식 문으로 구성됨
    - 대입문
    - 함수호출

```
perimeter = 2.0 * PI * radius;      // 대입문 (대입연산자 수식)
++counter;                          // 전위 증가 연산자 수식
counter--;                          // 후위 감소 연산자 수식
printf("Hello, World!");            // 함수 호출
```

빈 문장(empty statement)

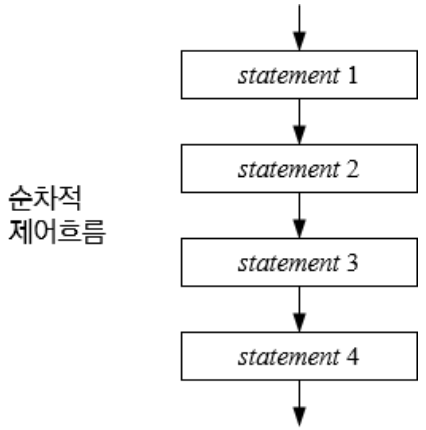
- 빈 문장(empty statement)
  - 실행할 것이 없는 문장
    - 수식문장에서 수식이 없는 경우
    - 문장의 끝을 표시하는 세미콜론은 반드시 표시해 주어야 함

```
;       // 빈 문장
{ }     // 빈 블록
{ ; }   // 빈 문장을 포함하고 있는 블록
```

```
// 스트링 str의 길이를 리턴하는 함수
size_t strlen(const char *str)
{
  const char *s;

  for (s = str; *s; ++s)
    ;                              // 빈 문장

  return (s - str);
}
```

제어 흐름

- 제어 흐름(Control flow)
  - 제어(control)
    - 프로그램은 여러 문장으로 구성됨
    - 프로그램 실행 시 어느 한 순간에 컴퓨터는 프로그램의 한 문장을 실행하고 있음
    - 이 시점에서 프로그램의 한 문장이 컴퓨터를 제어(control)하고 있음
    - 이 문장이 실행된 다음에는 그 다음에 오는 문장이 실행됨
  - 제어 흐름(control flow)
    - 프로그램을 구성하는 문장이 실행되는 순서
    - 순차적 제어 흐름
      - 작성된 문장의 순서대로 차례대로 실행됨



순차적
제어흐름

제어 흐름

- 제어 문장
  - 제어 흐름을 비순차적으로 만드는 문장

제어 문장	
선택문	if 문장 if-else 문장 switch 문장
반복문	for 문장 while 문장 do-while 문장
점프문	break 문장 continue 문장 return 문장

선택문

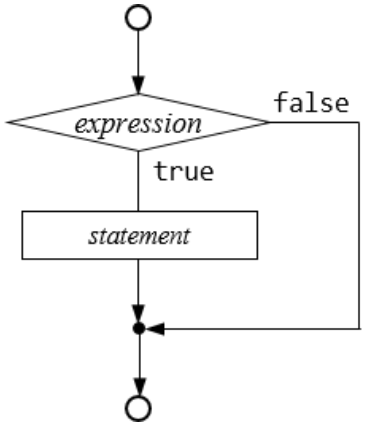
- 선택문(selection statement)
  - 조건문(conditional statement)라고도 부름
  - 프로그램이 실행되는 두 개 이상의 경로 중에서 한 경로를 선택할 수 있도록 함
  - 종류
    - `if` 선택문
    - `if-else` 선택문
    - `switch` 선택문

if 선택문

- if 선택문
  - 사용 문법

```
if ( expression ) statement
```

    - 수식 *expression* 의 결과
      - true : *statement* 를 실행
      - false : *statement* 가 실행되지 않음




```
if ( expression )
    statement
```

```
if ( expression ) {
    statement      // 블록
}
```

  - 예제

```
// 음수를 양수로 변환하는 if문
if (x < 0)
  x = -x;
```

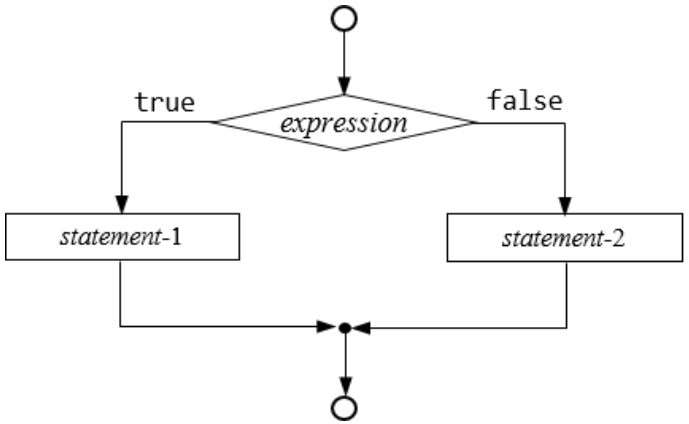
```
// 두 숫자 a, b 중에서 큰 수 구하기
max = a;
if (max < b)
  max = b;
```

if-else 선택문

- if-else 선택문
  - 사용 문법

```
if ( expression )
  statement-1
else
  statement-2
```

    - 수식 *expression* 의 결과
      - true : *statement*-1 을 실행
      - false : *statement*-2 를 실행




```
if ( expression )
   statement-1
else
   statement-2
```

```
if ( expression ) {
   statement-1    // 블록
}
else {
   statement-2    // 블록
}
```

  - 예제

```
// 두 숫자 a, b 중에서 큰 수 구하기
if (a >= b)
  max = a;
else
  max = b;
```

if-else 선택문

- if-else 선택문을 조건식으로 표현
  - 조건 연산자 ?: 를 사용하면 간단한 if-else 선택문을 대체할 수 있음

```
if (a > b)
  max = a;
else
  max = b;
```

⟺

```
int max = a > b ? a : b;
```

```
if (number % 2 == 0)
  isEven = true;
else
  isEven = false;
```

⟺

```
bool isEven = num % 2 == 0;
```

중첩 if 선택문

- 중첩(nested) if 선택문
  - if 선택문의 *statement*, if-else 선택문의 *statement*-1, *statement*-2 가
    - 또 다른 if 선택문 혹은 if-else 선택문이 될 수 있음
  - 내부의 if 선택문, if-else 선택문도 다시 중첩될 수 있음
    - 중첩의 깊이에는 한도가 없음
  - 예제

```
if (number > 0)
  cout << "number is positive" << endl;
else
{
  if (number < 0)
    cout << "number is negative" << endl;
  else
    cout << "number is zero" << endl;
}
```

```
if (number > 0)
  cout << "number is positive" << endl;
else if (number < 0)
  cout << "number is negative" << endl;
else
  cout << "number is zero" << endl;
```

중첩 if 선택문

- 중첩(nested) if 선택문
  - 예제

```
if (score >= 90)
    grade = 'A';
else
    if (score >= 80)
        grade = 'B';
    else
        if (score >= 70)
            grade = 'C';
        else
            if (score >= 60)
                grade = 'D';
            else
                grade = 'E';
```

동일한 코드

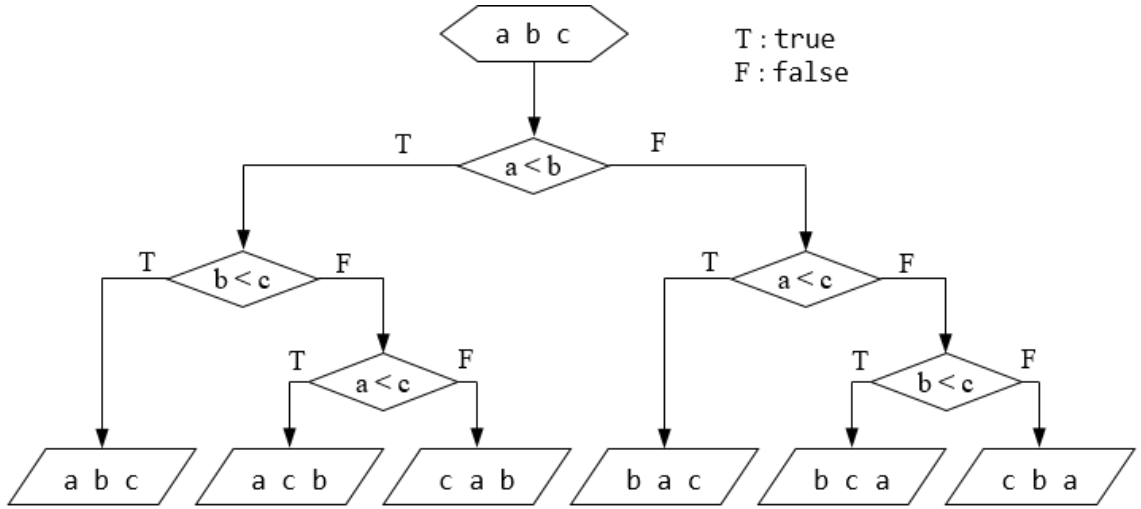
더 좋은 코드

```
if (score >= 90)
    grade = 'A';
else if (score >= 80)
    grade = 'B';
else if (score >= 70)
    grade = 'C';
else if (score >= 60)
    grade = 'D';
else
    grade = 'E';
```

중첩 if 선택문

- 중첩(nested) if 선택문
  - 예제: 세 숫자 a b c 의 정렬

```
if (a < b)
  if (b < c)
    cout << a << " " << b << " " << c << endl;
  else if (a < c)
    cout << a << " " << c << " " << b << endl;
  else
    cout << c << " " << a << " " << b << endl;
else
  if (a < c)
    cout << b << " " << a << " " << c << endl;
  else if (b < c)
    cout << b << " " << c << " " << a << endl;
  else
    cout << c << " " << b << " " << a << endl;
```



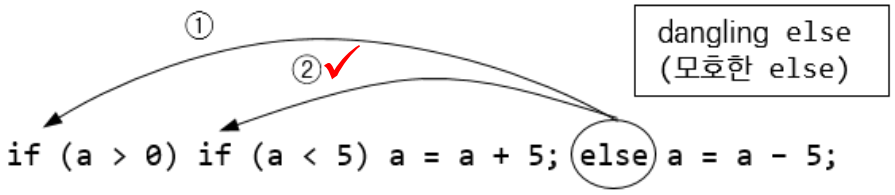
모호한 else

- 모호한 else (dangling else)
  - 다음 코드의 결과는?

```
a = 7;
if (a > 0) if (a < 5) a = a + 5; else a = a - 5;
cout << a << endl;
```

2

  - else가 연결되는 if는?
    - 다른 else와 연결되어 있지 않은 가장 가까운 if와 연결함



①

```
if (a > 0) {
  if (a < 5)
    a = a + 5;
}
else
  a = a - 5;
```

② ✓

```
if (a > 0) {
  if (a < 5)
    a = a + 5;
  else
    a = a - 5;
}
```

모호한 else

- 모호한 else (dangling else)
  - 다음 코드의 결과는?

ⓐ
```
a = 7;
if (a > 0)
  if (a < 5)
    a = a + 5;
else
  a = a - 5;
cout << a << endl;
```

ⓑ
```
a = 7;
if (a > 0)
  if (a < 5)
    a = a + 5;
  else
    a = a - 5;
cout << a << endl;
```

- 두 경우 모두 2가 출력됨
- 들여쓰기(indentation)와 무관함
- else를 첫 번째 if 와 연결시키기 위해서는 두 번째 if 문을 블록으로 만들어야 함

```
if (a > 0) {
  if (a < 5)
    a = a + 5;
}
else
  a = a - 5;
```

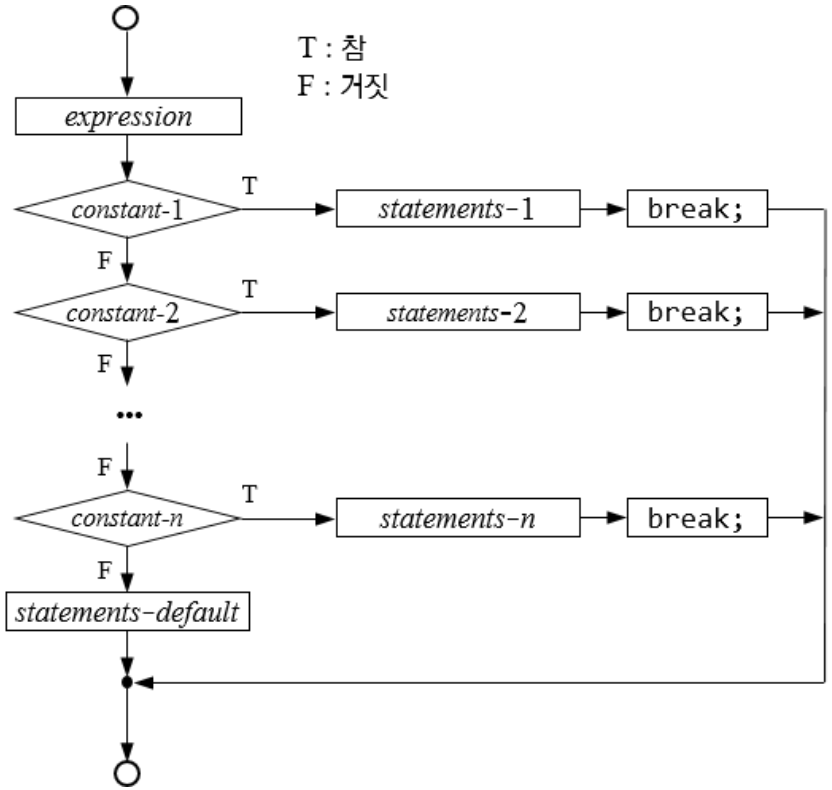
switch 문장

- switch 문
  - 기능
    - 다중 분기(multiway branch) 제공
    - 여러 개의 항목 중에서 한 개를 선택할 수 있도록 함
    - 중첩된 if 문장보다는 효과적인 방법임

switch 문장

- switch 문
  - 사용 문법

```
switch(expression)              // 수식 expression
{
    case constant-1:            // case label
        statements-1
        break;
    case constant-2:            // case label
        statements-2
        break;
.
.
.
    case constant-n:            // case label
        statements-n
        break;
    default:                    // default label
        statements-default
};
```



- *expression* : 정수 혹은 열거형 결과값
- *constant-** : *expression* 과 같은 자료형의 상수, 상수 수식

switch 문장

- switch 문
  - 사용 문법
    - *expression* : 정수 혹은 열거형 결과값
    - *constant-** : *expression* 과 같은 자료형의 상수, 상수 수식
      - 변수를 포함하는 수식은 안됨
      - 상수들은 모두 다른 값이어야 함
      - 중복으로 상수가 나타날 경우에는 컴파일러 오류가 발생함
    - default 레이블은 없을 수도 있음
    - 각 case 레이블과 default 레이블이 나열되는 순서는 중요하지 않음
  - 실행 과정
    - 수식 *expression* 을 계산함
    - 그 결과값과 case 레이블의 상수들을 순서대로 비교함
    - 첫번째로 일치하는 상수에 해당하는 case의 문장 *statements*를 실행함
    - 그 다음에 오는 break문을 실행하여 switch문의 실행을 종료함
    - 일치하는 상수가 없을 경우에는 default 레이블의 문장 *statement-default*를 실행하고 switch문의 실행을 종료함

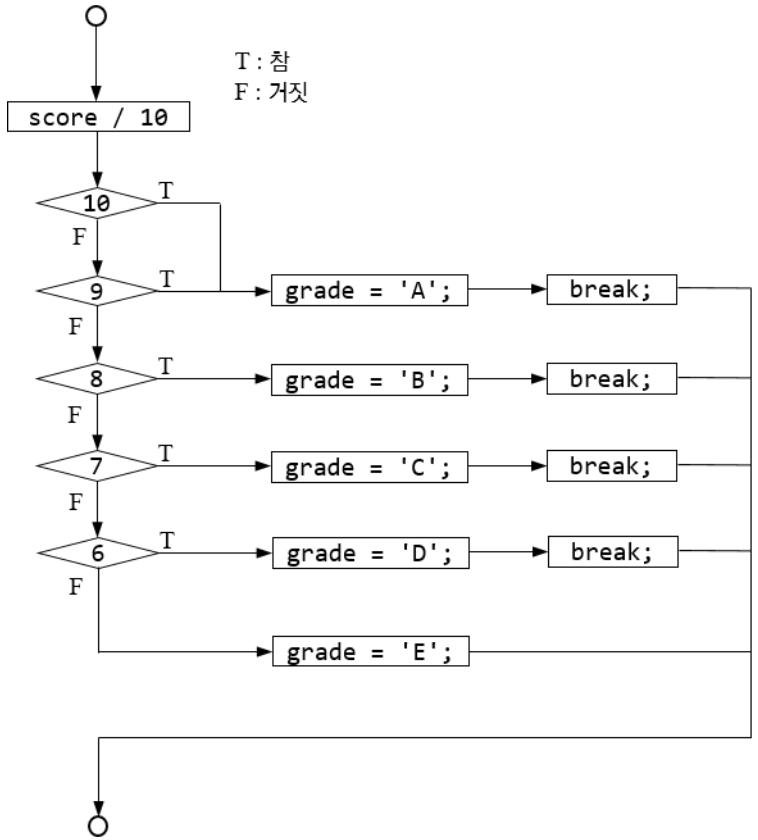
```
switch(expression)              // 수식 expression
{
    case constant-1:            // case label
        statements-1
        break;
    case constant-2:            // case label
        statements-2
        break;
.
.
.
    case constant-n:            // case label
        statements-n
        break;
    default:                    // default label
        statements-default
};
```

switch 문장

- 예제
  - 점수에 따른 학점 계산

```
if (score >= 90)
    grade = 'A';
else if (score >= 80)
    grade = 'B';
else if (score >= 70)
    grade = 'C';
else if (score >= 60)
    grade = 'D';
else
    grade = 'E';
```

```
switch (score/10)
{
  case 10:
            grade = 'A';
            break;
  case 9:
            grade = 'A';
            break;
  case 8:
            grade = 'B';
            break;
  case 7:
            grade = 'C';
            break;
  case 6:
            grade = 'D';
            break
  default:
            grade = 'E';
};
```

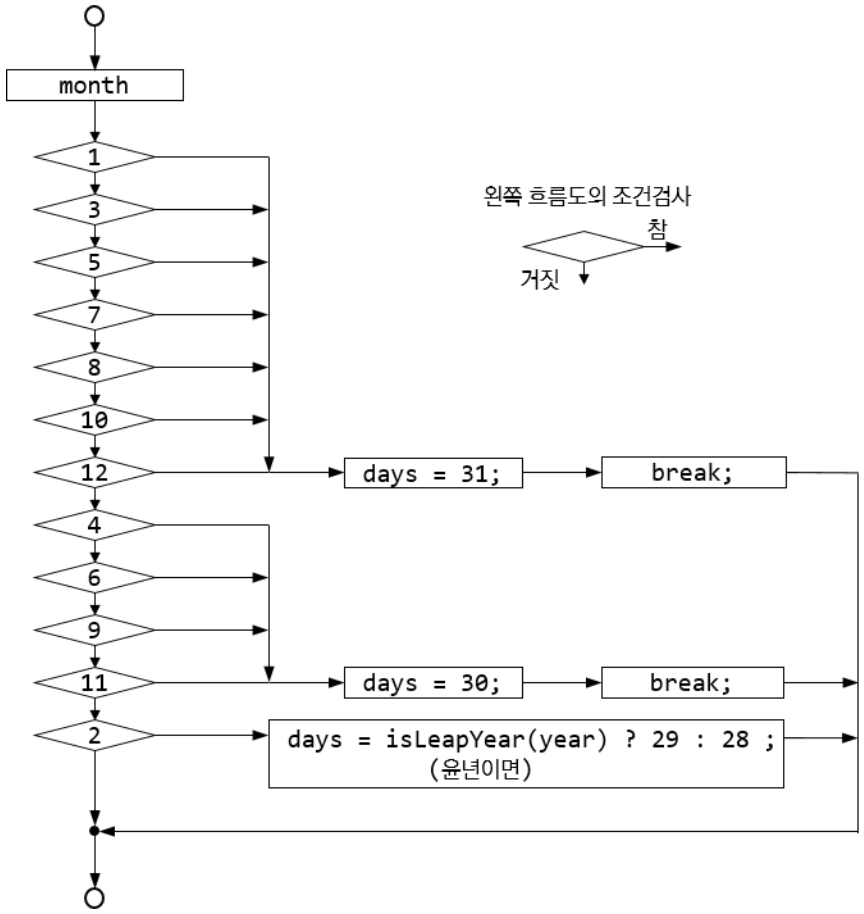



```
switch(score/10)
{
  case 10:
  case 9:   grade = 'A'; break;
  case 8:   grade = 'B'; break;
  case 7:   grade = 'C'; break;
  case 6:   grade = 'D'; break
  default:
            grade = 'E';
};
```

switch 문장

- 각 case 레이블의 마지막 문장 break의 역할
  - switch문의 실행을 종료하는 역할
  - 문장 break가 없을 경우에는 낙하적 흐름(fall through)으로 다음 문장들이 실행됨
    - 다음 break문장을 만나거나 switch문장의 끝을 만날 때까지 순차적으로 문장을 실행함
- 예제
  - 지정된 년도의 월에 속하는 날짜의 수를 계산

```
switch(month)
{
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12: days = 31;
           break;
  case 4:
  case 6:
  case 9:
  case 11: days = 30;
           break;
  case 2:
           days = isLeapYear(year) ? 29 : 28;
};
```



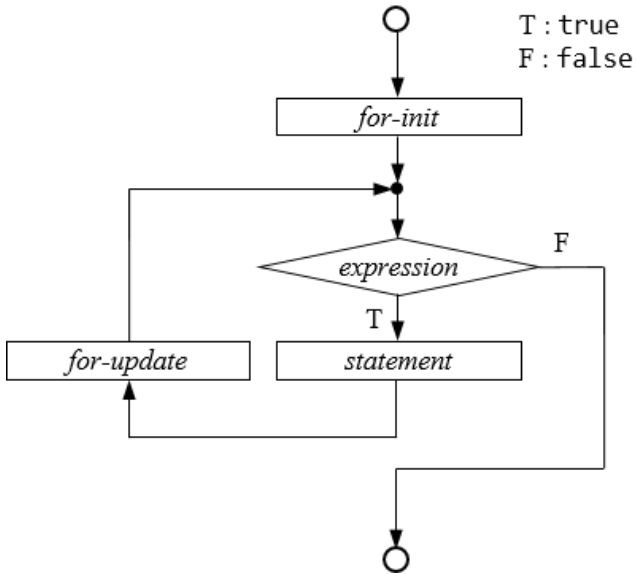
반복문

- 반복문(iteration statement)
  - 고정된 횟수만큼 반복 실행
  - 주어진 조건이 만족하는 동안 계속 반복 실행
  - 종류
    - `for` 반복문
    - `while` 반복문
    - `do-while` 반복문

for 반복문

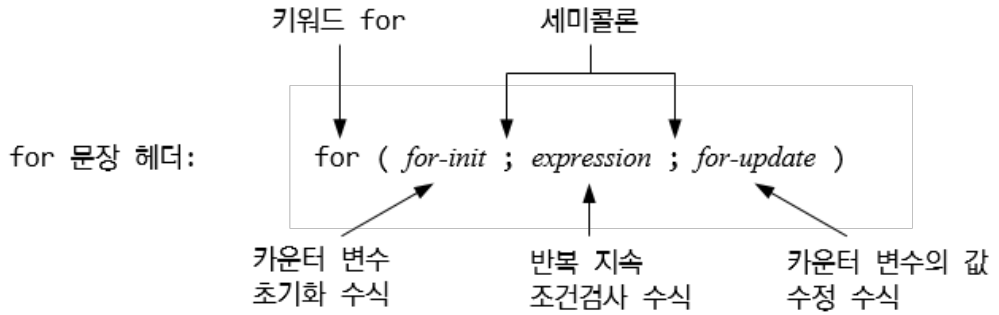
- for 반복문
  - 미리 반복 횟수를 알고 있는 경우에 주로 사용하는 반복문
  - 반복 횟수를 계산하는 카운터(counter) 변수를 사용하여 반복을 제어
    - 주로 많이 사용하는 카운터 변수명: i, j, k
  - 사용 문법

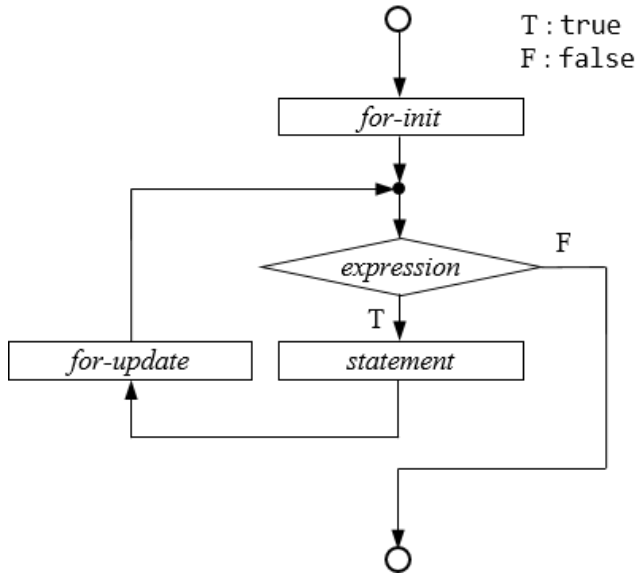
```
for ( for-init ; expression ; for-update )
    statement
```



for 반복문

- 실행 과정
  - *for-init*
    - 이 부분을 먼저 실행함 (오직 한 번만 실행)
    - 수식 문장 혹은 선언문
      - » 수식 문장: 카운터 변수를 초기화하는 수식
      - » 선언문 : 카운터 변수를 선언하면서 초기화함
  - *expression*
    - 반복문의 몸체인 *statement*를 실행하기 이전에 수식을 계산함
    - 수식의 계산 결과
      - » true : 반복문의 몸체인 *statement*를 실행함
      - » false: 반복문이 종료됨
  - *for-update*
    - 반복문의 몸체인 *statement*를 실행한 후에 계산하는 수식
    - 그 다음에는 *expression* 을 다시 계산함
    - 주로 카운터 변수의 값을 수정하는 수식임
  - *statement*
    - 반복문의 몸체(주로 복합문임)



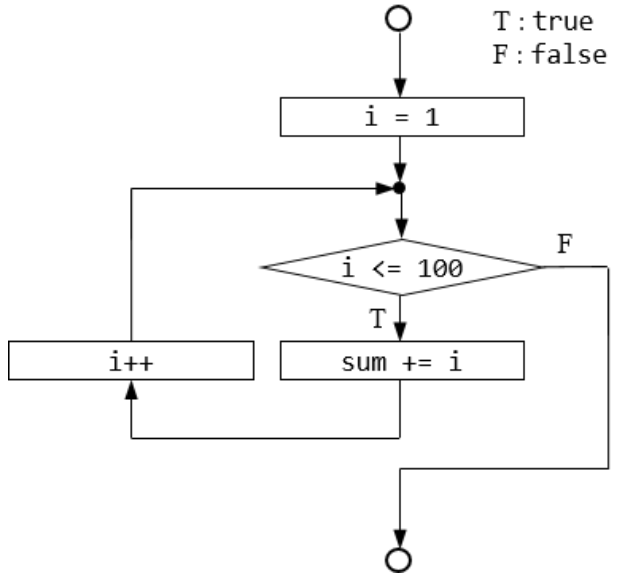


for 반복문

- 예제:
  - 1부터 100까지의 모든 정수의 합을 계산

반복 몸체

```
int sum = 0;
int i;

for (i = 1 ; i <= 100 ; i++)
    sum += i;

cout << "The sum is " << sum << endl;
```



  - 1부터 100까지의 정수 중에서 5의 배수의 합을 계산

```
int sum = 0;

for (int i = 1 ; i <= 100 ; i += 5)
    sum += i;

cout << "The sum is " << sum << endl;
```

for 반복문

- 예제:
  - 1부터 100까지의 모든 짝수의 합을 계산

```
int sum = 0;

for (int i = 100; i > 0; i -= 2)
    sum += i;

cout << "The sum is " << sum << endl;
```

for 반복문

- *for-init* 부분에서 선언된 카운터 변수
  - 카운터 변수는 for 반복문에서만 사용할 수 있음
  - for 반복문을 벗어나면 사용할 수 없음

```
// for-init 부분에서 카운터 변수 선언 및 초기화
for (int i = 0 ; i <= 100 ; i++)
    sum += i;

// for 반복문 다음에서는 변수 i를 사용할 수 없음
```

- *for-init* 부분은 생략할 수 있음

```
// 카운터 변수 선언 및 초기화
int i = 0;

for ( ; i <= 100 ; i++) // for-init 부분이 생략됨
    sum += i;
```

for 반복문

- 무한 반복(infinite loop)
  - for 반복문의 *expression* 부분이 생략
    - 조건식이 항상 true인 것으로 간주하여 무반 반복이 이루어짐
  - for 반복문의 몸체에서 반복문을 끝낼 수 있은 break 문이 있어야 함

```
// 무한 반복 for문
for ( ; ; ) {

   // 내부에 반복을 종료할 break문이 있어야 함.
}
```

```
// 무한 반복 for문
for ( ; true ; ) {

   // 내부에 반복을 종료할 break문이 있어야 함.
}
```

- 한 개 이상의 카운터 변수를 사용할 수 있음
  - 콤마 연산자 사용

```
int i, j;

for (i = 0, j = 1 ; ; i++, j += 2) {
  ...
}
```

```
for (int i = 0, j = 1 ; ; i++, j += 2) {
  ...
}
```

for 반복문

- 예제:
  - 어떤 해의 1월 달력을 출력하는 프로그램
    - 1월 1일의 요일은 변수로 주어짐

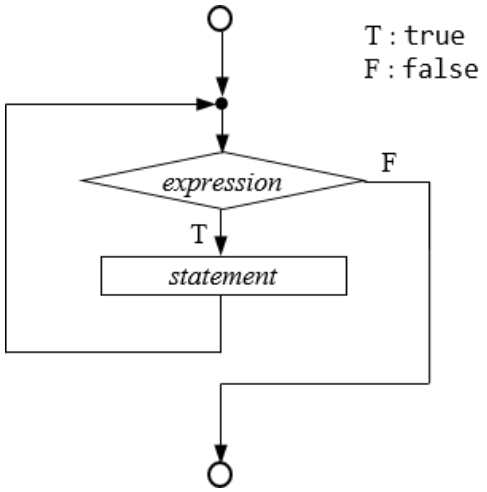
```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    int date = 4;  // 1월 1일의 요일 (0: 일요일, 1: 월요일, …, 6: 토요일)
    int col;       // 현재 날짜를 출력할 열(요일별 열)

    cout << "Sun Mon Tue Wed Thu Fri Sat" << endl;
    cout << "--- --- --- --- --- --- ---" << endl;

    col = date;
    for (int i = 0; i < col; i++)  // 1일을 출력하기 전에 공백을 먼저 출력함
        cout << "    ";

    for (int day = 1; day <= 31; day++) {
        cout << setw(3) << day;
        col++;
        if (col <= 6)
            cout << " ";
        else if (day < 31) {
            cout << endl;
            col = 0;
        }
    }
    return 0;
}
```

```
Sun Mon Tue Wed Thu Fri Sat
--- --- --- --- --- --- ---
                  1   2   3
  4   5   6   7   8   9  10
 11  12  13  14  15  16  17
 18  19  20  21  22  23  24
 25  26  27  28  29  30  31
```

while 반복문

- while 반복문
  - 어떤 조건을 만족하는 동안에 주어진 문장을 반복 실행
  - 사용 문법

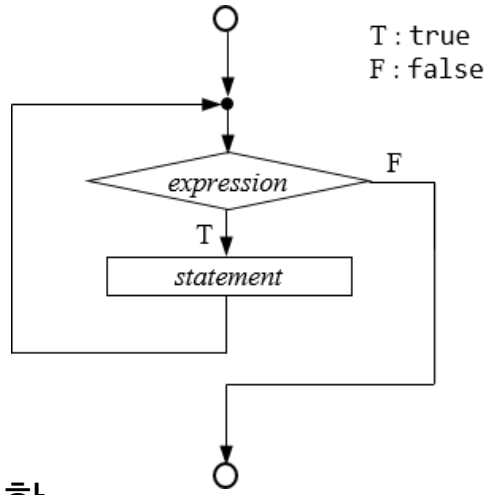
```
while ( expression )
    statement
```



while 반복문

- 실행 과정
  - *expression*
    - 이 수식을 먼저 계산함
    - 수식의 계산 결과
      - » true : 반복문의 몸체인 *statement*를 실행함
      - » false: 반복문이 종료됨
  - *statement*
    - 반복문의 몸체(주로 복합문임)
    - 그 다음에는 *expression* 을 다시 계산함

```
while ( expression )
    statement
```



- 참고사항
  - 수식 *expression* 을 처음 계산한 결과가 false이면
    - statement가 한 번도 실행되지 않음
  - 몸체인 *statement* 에서는 변수 값을 변경하여 *expression* 을 false로 만들어야 함
    - 그렇지 않으면 무한 반복이 실행됨

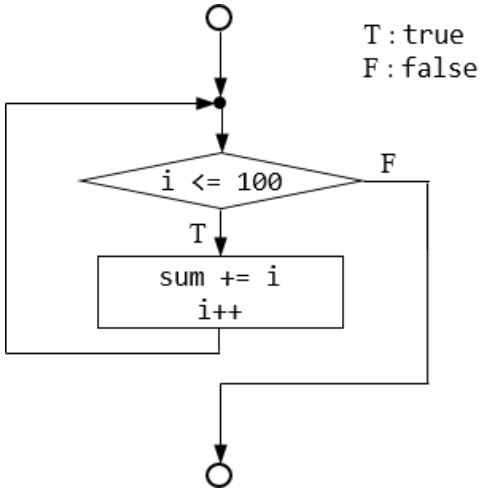
while 반복문

- 예제:
  - 1부터 100까지의 모든 정수의 합을 계산

수식

반복 몸체

```
int sum = 0;
int i = 1;

while ( i <= 100 ) {
    sum += i;
    i++;
}

cout << "The sum is " << sum << endl;
```



  - 1부터 100까지의 정수 중에서 5의 배수의 합을 계산

```
int sum = 0;
int i = 0;

while ( i <= 100 ) {
    sum += i;
    i += 5;
}

cout << "The sum is " << sum << endl;
```

while 반복문

- 무한 반복(infinite loop)
  - while 반복문의 *expression* 부분이 항상 true인 경우
    - *expression* 부분은 for 반복문에서와 같이 생략할 수는 없음
      - » 생략되면 컴파일러 오류가 발생함
  - while 반복문의 몸체에서 반복문을 끝낼 수 있은 break 문이 있어야 함

```
// 무한 반복 while문
while ( true ) {

   // 내부에 반복을 종료할 break문이 있어야 함.
}
```

```
// 무한 반복 while문
while ( ) {  // 컴파일러 오류:
             // 수식 expression은 생략할 수 없음

}
```

while 반복문

- 예제:
  - 예금의 원금과 금리가 주어졌을 때, 복리예금을 하였을 때 몇 년이 지나야 원금의 두 배가 되는가?

$$P_n = P(1+r)^n$$

P_n : n년 후의 예금액
P : 원금
r : 년 이자율
n : 년

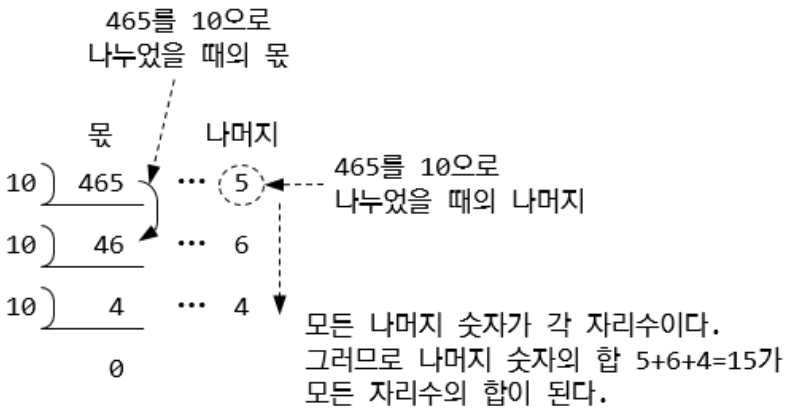
```
#include <iostream>
using namespace std;

int main()
{
    double intrestRate = 0.05; // 예금 금리: 년 이자율 5%
    double principal = 1.0;    // 원금(=1.0)
    int year = 0;              // 매년 복리로 늘어나는 원금을 계산함

    while (principal < 2.0) {          // 원금이 2배가 되면 반복문을 종료함
        principal *= 1 + intrestRate;  // 복리로 원금이 계속 늘어남
        year++;
    }

    cout << year << "년 후에 원금이 두 배가 됩니다." << endl;
    return 0;
}
```

```
15년 후에 원금이 두 배가 됩니다.
```

while 반복문

- 예제:
  - 주어진 정수 num의 모든 자리수의 합을 계산하는 프로그램
    - 예

465	모든 자리수의 합: 4+6+5=15



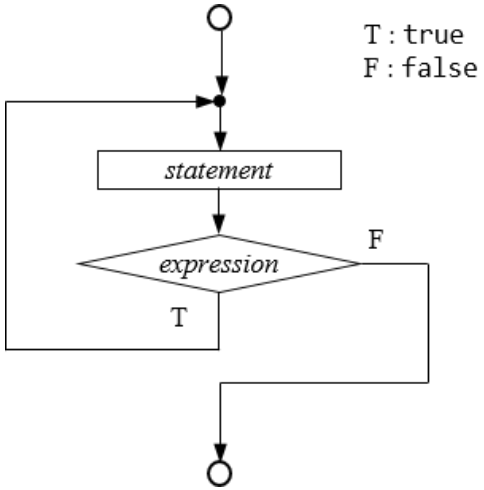

```
#include <iostream>
#include <climits>
using namespace std;

int main()
{
    unsigned long long int num = ULLONG_MAX;
    int sumDigits = 0;    // 정수 num의 모든 자리수의 합

    cout << "ULLONG_MAX is " << num << endl;
    while (num)
    {
        sumDigits += num % 10; // 자리수(num을 10으로 나누었을 때의 나머지)를 더함
        num /= 10;             // num을 10으로 나누었을 때의 몫
    }
    cout << "Sum of digits of ULLONG_MAX is " << sumDigits << endl;

    return 0;
}
```

```
ULLONG_MAX is 18446744073709551615
Sum of digits of ULLONG_MAX is 87
```

do-while 반복문

- do-while 반복문
  - while 반복문과 유사함
    - 반복 몸체를 먼저 실행
    - 그 다음에 반복을 계속해야 할지를 결정하는 조건을 검사함.
  - 사용 문법

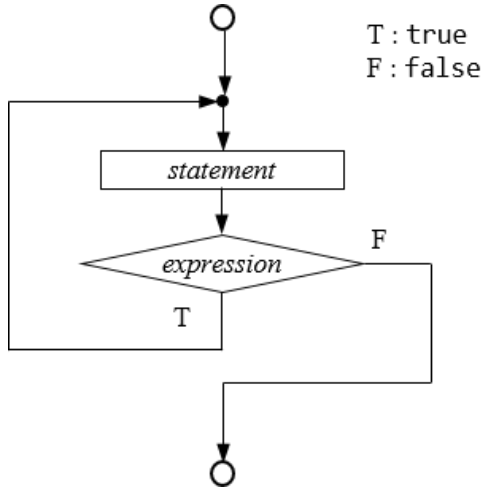
```
do
    statement
while ( expression );
```



do-while 반복문

- 실행 과정
  - *statement*
    - 반복문의 몸체(주로 복합문임)
    - 그 다음에 *expression* 을 계산함
  - *expression*
    - 반복을 계속해야 하는지 혹은 종료해야 하는지를 판단하는 조건 수식
    - 수식의 계산 결과
      - » true : 반복문의 몸체인 *statement*를 실행함
      - » false: 반복문이 종료됨

```
do
      statement
while ( expression );
```

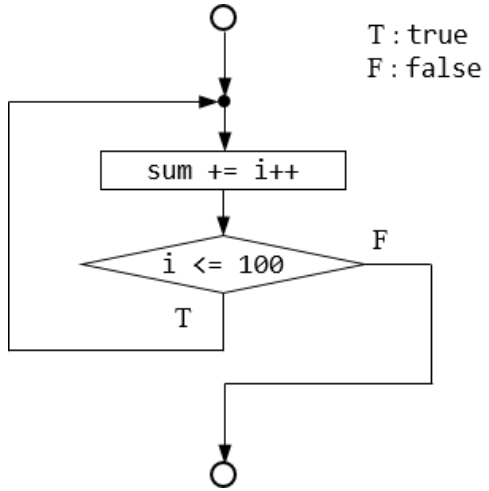


- 참고사항
  - 반복 몸체인 *statement* 가 최소 한 번 실행됨
    - `while` 반복문에서는 한 번도 실행되지 않을 수 있음
  - 몸체인 *statement* 에서는 변수 값을 변경하여 *expression* 을 false로 만들어야 함
    - 그렇지 않으면 무한 반복이 실행됨

do-while 반복문

- 예제:
  - 1부터 100까지의 모든 정수의 합을 계산

```
int sum = 0;
int i = 1;

do
   sum += i++;
while (i <= 100);

cout << "The sum is " << sum << endl;
```



do-while 반복문

- 예제: 최대공약수 계산(유클리드 호제법)
  - 최대공약수 GCD(Greatest Common Divisor)
    - $\gcd(a, b)$: 두 개의 자연수 a, b $(a > b)$ 가 주어졌을 때, a와 b의 약수 중에서 가장 큰 약수
    - 예: $\gcd(630, 336) = 42$
  - 유클리드 호제법(Euclidean algorithm)
    - 최대공약수를 계산하는 알고리즘 중의 하나
    - 호제법(互除法)이란 두 수가 서로(互) 상대방의 수를 나누면서(除) 알고리즘이 진행된다는 것을 나타냄

do-while 반복문

– 예제: 최대공약수 계산(유클리드 호제법)

유클리드 호제법:

두 자연수 a, b $(a > b)$에 대하여, a를 b로 나눈 나머지를 r $(r < b)$이라고 하면, a와 b의 최대공약수는 b와 r의 최대공약수와 같다. 즉

$$\gcd(a, b) = \gcd(b, r)$$

만약 $r = 0$이라면, a와 b의 최대공약수는 b이다.

```
a = 630; b = 336;

do {
    r = a % b;      // a를 b로 나눈 나머지
    a = b;
    b = r;
} while (r > 0);

cout << "최대공약수: " << a << endl;
```

```
최대공약수: 42
```

```
  gcd(630,336)
= gcd(336,294)
= gcd(294,42)
= gcd(42,0)
= 42
```