

C 프로그래밍 출석수업

컴퓨터과학과 이병래교수





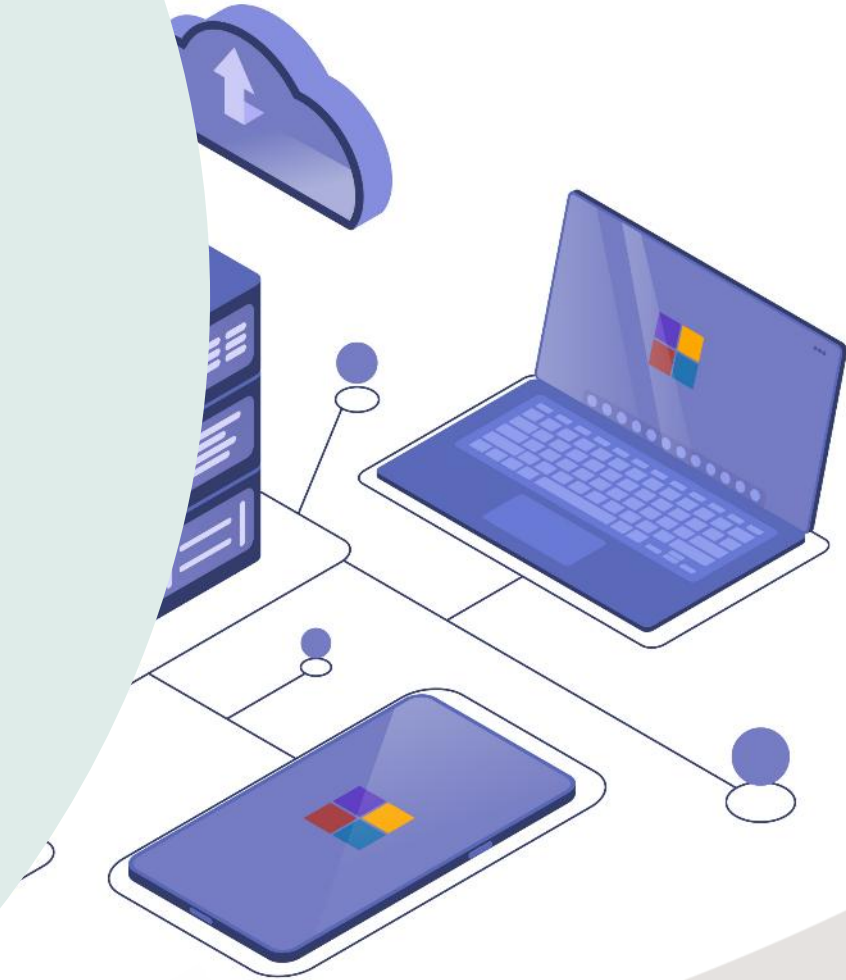
학습목차

- 1 C 프로그램의 기초
- 2 자료형과 상수 및 변수
- 3 연산자
- 4 흐름제어



C 프로그래밍 출석수업

1 C 프로그램의 기초



(1) C 프로그램의 기본 구조

파일명: □□□□.c

```
/* 도입부 */  
.....
```

```
/* main 함수 */  
int main()  
{  
    /* 선언문, 명령문 등 */  
    .....  
}
```

```
/* 프로그램에 필요한 함수 선언 */  
void f()  
{  
    .....  
}
```



소스코드의 확장자: '.c'



반드시 하나 이상의 함수를 포함해야 한다.

- main() 함수가 반드시 존재해야 한다.
- 함수의 몸체는 시작과 끝을 알리는 중괄호 {}를 사용하여 블록으로 구성한다.
- 블록 안에는 선언문, 치환·연산·함수호출 등의 문장을 기입한다.



선행처리 지시어(preprocessing directives)를 제외한 문장의 끝에는 세미콜론(;)을 붙인다.



모든 문장은 문법과 의미에 맞게 '토큰(token)'을 나열하여 작성한다.

(1) C 프로그램의 기본 구조

■ 예: “Hello, World” 프로그램

파일명: HelloWorld.c

```
/* 도입부 */  
#include <stdio.h>
```

```
/* main 함수 */  
int main()  
{  
    printf("Hello, World!\n");  
}
```

- ✎ 소스코드의 확장자: '.c'
- ✎ 반드시 하나 이상의 함수를 포함해야 한다.
 - main() 함수가 반드시 존재해야 한다.
 - 함수의 몸체는 시작과 끝을 알리는 중괄호 {}를 사용하여 블록으로 구성한다.
 - 블록 안에는 선언문, 치환·연산·함수호출 등의 문장을 기입한다.
- ✎ 선행처리 지시어(preprocessing directives)를 제외한 문장의 끝에는 세미콜론(;)을 붙인다.
- ✎ 모든 문장은 문법과 의미에 맞게 '토큰(token)'을 나열하여 작성한다.

(2) C 프로그램의 토큰

■ 토큰(token): 소스 코드에서 의미를 갖는 최소 단위의 단어나 기호

■ C 프로그램을 구성하는 토큰의 종류

- 예약어(키워드): C에서 고유한 문법 및 의미가 정해진 단어
- 명칭: 변수, 함수 등을 식별하기 위해 정의하는 이름
- 상수: 값이 변하지 않는 자료(정수, 실수, 문자 등)
- 문자열: 큰따옴표로 묶인 문자 시퀀스
- 구두점: 고유한 문법 및 의미가 정해진 기호
 - 예: `=`, `+`, `-`, `*`, `/` 등의 연산자나 `;`, `{`, `}`, 괄호 등의 구분자
- 설명문: 프로그램에 대한 주석
 - `/*` `*/` 또는 `//`

(2) C 프로그램의 토큰

■ 예약어(reserved word, keyword)

- C에서 고유한 문법 및 의미가 정해진 단어
 - 정해진 용도가 아닌 다른 용도로는 사용할 수 없음
- 예약어의 종류
 - 자료형 관련 예약어: char, int, float, short, long, double, unsigned, struct, union, enum, void, typedef 등
 - 기억 관련 예약어: auto, static, extern, register, volatile, const, sizeof 등
 - 제어 관련 예약어: if, else, switch, case, default, for, while, do, break, continue, return 등

(2) C 프로그램의 토큰

■ 명칭(identifier)

- 프로그램 내의 여러가지 요소를 식별하기 위해 정의함
 - 변수, 함수 등
- 명칭 정의 규칙
 - 영문자(대·소문자 구분함)와 숫자의 조합으로 만든다.
 - 명칭의 첫 문자는 영문자나 밑줄('_')이어야 한다.
 - 특수문자를 사용할 수 없다. 단, 밑줄은 사용할 수 있다.
 - 문자 사이에 공백이 있어서는 안 된다.
 - 예약어를 사용할 수 없다.

(2) C 프로그램의 토큰

■ 명칭(identifier)

● 올바르게 정의한 명칭의 예

올바른 명칭	비 고
myname	영문자로 구성된 명칭
myName	myname과는 다른 명칭(Camel case)
MyName	myname과는 다른 명칭(Pascal case)
my_name	밑줄 사용 가능(snake case)
flag01	영문자와 숫자 조합 가능
For	사용 가능(예약어 for와는 구분됨)

(2) C 프로그램의 토큰

■ 명칭(identifier)

● 잘못된 명칭의 예

잘못된 명칭	비 고
my name	문자 사이 공백 사용 불가
my#name	특수문자 사용 불가
my - name	특수문자 사용 불가
4days	숫자로 시작 불가
auto	예약어 사용 불가

(2) C 프로그램의 토큰

■ 상수(constant)

- 수치 상수, 문자 상수, 문자열 상수

예

123 'a' "KNOU"

■ 연산자(operator)

- 여러 가지 유형의 연산을 표현함
 - 산술 연산, 관계 연산, 논리 연산, 비트 연산, 대입 연산 등
- 연산의 대상을 피연산자(operand)라고 함
 - 연산자의 종류에 따라 1~3개의 피연산자가 사용됨

(2) C 프로그램의 토큰

■ 설명문(comment, 주석문)

- 프로그램에 대한 설명을 작성하는 문장
 - 임의의 형식으로 작성하며, 컴파일러는 설명문의 영역은 무시함
- 작성 방법
 - ‘/*’와 ‘*/’ 사이에 작성한 모든 문장은 설명문임
 - ▶ 여러 줄에 걸친 긴 설명문을 작성할 때 편리함
 - ‘//’ 기호 이후의 문장은 그 행의 끝까지 설명문임
 - ▶ 간단한 설명문을 작성할 때 편리함

(2) C 프로그램의 토큰

■ 설명문(comment, 주석문)

예

```
/*   콘솔에 메시지를 출력하는  
    간단한 C 프로그램   */  
#include <stdio.h>  
int main() {  
    // 메시지 출력 출력  
    printf("Hello, World!\n");  
}
```

(3) 변수

■ 변수란?

- 프로그램에서 값을 저장하기 위한 기억공간
- 프로그램 실행 도중 값을 변경할 수 있음
- 모든 변수는 사용하기 전에 미리 선언되어야 함

■ 변수의 특징

- 모든 변수는 이름이 있다.
- 모든 변수는 정해진 자료형이 있다.
- 모든 변수는 값을 갖는다.

(3) 변수

■ 변수의 선언

형식

```
자료형 변수명;  
자료형 변수명1, 변수명2;
```

예

```
int    a;  
double b;  
a = 3 + 2;  
b = a / 4.0;
```

5

a

1.25

b

(4) C 프로그램의 문장

■ 문장(statement)

- 선언이나 실행을 지시하는 표현의 기본 단위
 - 선언문: 자료형, 변수 등을 선언하는 문장
 - ▶ 예: `int a;`
 - 실행문: 동작을 수행하도록 지시하는 문장
 - ▶ 예: `a += 10;`
 - ▶ 실행문은 함수 안에만 작성할 수 있음
- 모든 문장은 세미콜론(;)으로 끝을 표시함
- 문장의 실행은 나열된 순서에 따라 순차적으로 실행됨
 - 필요에 따라 흐름제어 문장을 사용하여 진행 방향을 바꿀 수 있음

(4) C 프로그램의 문장

■ 문장(statement)

● 식(expression)

- 순서에 맞게 나열한 연산자와 피연산자의 시퀀스

예

```
int x = 10, y = 20;    // 변수의 선언문
x + y;                // x와 y의 합을 계산하는 식(경고 발생)
x = y * 2;           // y에 2를 곱한 값을 x에 저장하는 식
```

- 선택 제어문: 조건에 따라 문장을 선택하여 실행함
- 반복 제어문: 조건에 따라 문장을 반복하여 실행함

(4) C 프로그램의 문장

■ 문장(statement)

- 복합문(compound statement, 블록, block)
 - 한 개 이상의 문장을 중괄호 { }로 묶어 놓아 하나의 문장처럼 취급할 수 있게 한 것
 - 내부에 지역변수를 포함할 수 있음
 - ‘}’로 복합문의 끝을 표시하므로 ‘;’으로 끝마칠 필요 없음

예

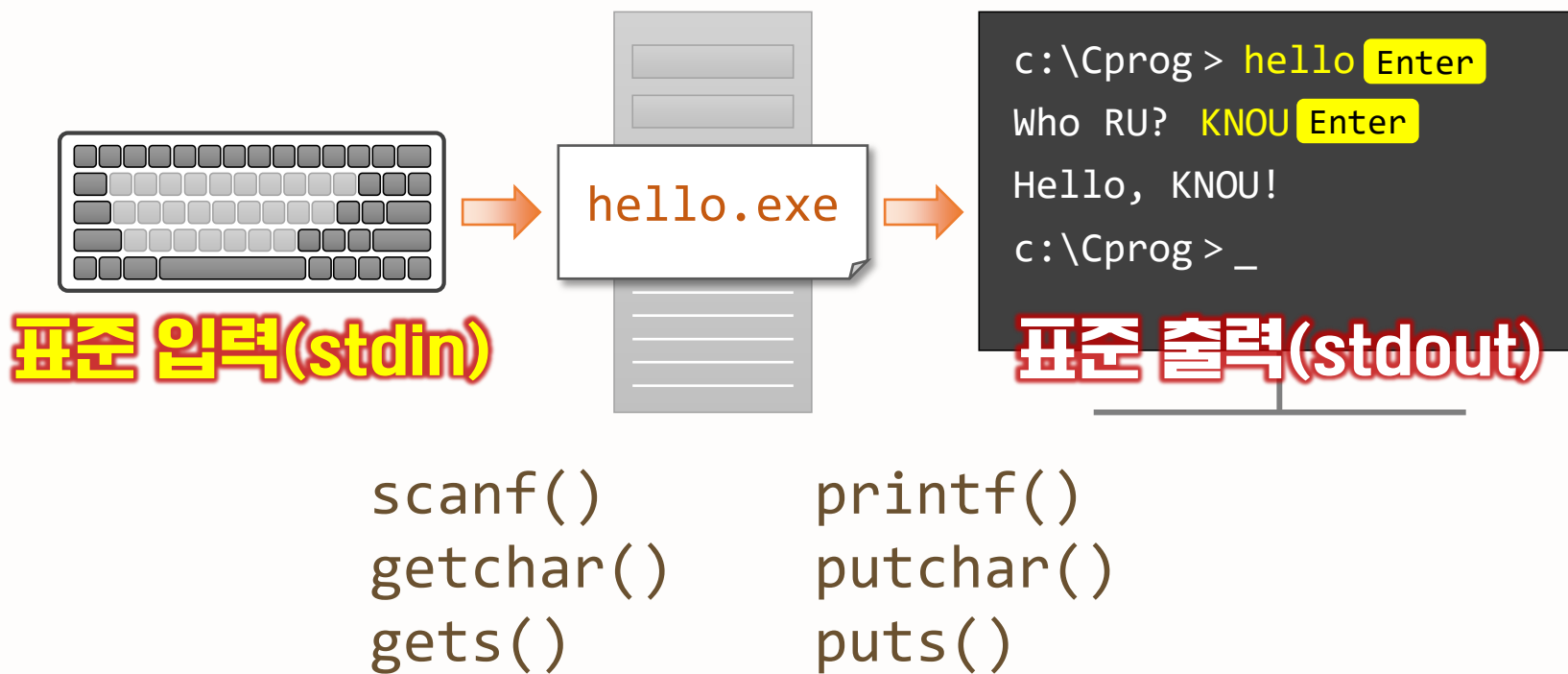
```
if (a > b) { // a와 b는 int형 변수
    int tmp = a; // 세 문장을 묶은 복합문을
    a = b;       // 조건이 참일 때 실행할
    b = tmp;     // 하나의 문장처럼 취급함
}
```

(5) 입력과 출력의 기초

■ 표준 입출력(standard input/output)

- 프로그램이 사용자와 데이터를 주고받는 기본적인 입력과 출력 장치

📝 데이터의 형태: 문자 시퀀스



(5) 입력과 출력의 기초

■ printf()

- 여러 종류의 자료를 지정된 양식으로 콘솔 화면에 출력하는 함수

형식

```
printf("출력양식");  
printf("출력양식", 식1[, 식2, ...]);
```



"출력양식": 출력할 내용을 문자열 형태로 표현

- 식이 이어지는 경우 식의 값을 문자열 형태로 변환하기 위한 양식 변환기호를 포함함



식에는 문자, 정수, 실수 등의 값을 내는 변수, 계산식, 함수호출 등을 사용할 수 있음

(5) 입력과 출력의 기초

■ printf()

```
1 #include <stdio.h>
2 int main() {
3     char c = 'A';
4     int i = 10, j = 20, k = 30;
5     printf("간단한 출력 프로그램\n");
6     printf("%c의 ASCII 코드값은 %d\n", c, c);
7     printf("i=%d, j=%d, k=%d\n", i, j, k);
8 }
```

출력 양식 변환기호

간단한 출력 프로그램

'A'의 ASCII 코드값은 65

i=10, j=20, k=30

(5) 입력과 출력의 기초

■ scanf() 함수

- 지정된 양식으로 값을 입력하여 변수(기억공간)에 저장함

형식

```
scanf("입력양식", &변수1[, &변수2, ...]);
```



"입력양식"에 포함되는 내용

- 빈칸 또는 탭 문자
- 양식 변환기호: 입력 문자열을 자료형에 맞는 값으로 변환하는 양식을 지정하기 위해 사용하는 '%'로 시작하는 기호
- 양식 변환기호나 백색공백이 아닌 문자: 입력과 맞아야 함



변수: 값을 저장할 변수의 주소가 필요하므로
주소 연산자 '&'를 사용 함

(5) 입력과 출력의 기초

■ scanf() 함수

● 예: 정수와 실수의 입력

```
1 #include <stdio.h>
2 int main() {
3     int a;
4     float f;
5     scanf("%d%f", &a, &f);
6     printf("%d %f\n", a, f);
7 }
```

입력 양식 변환기호

10 12.3 Enter

10 12.300000

(5) 입력과 출력의 기초

- [예제 1] int형의 값을 두 개 입력하여 이들의 합과 곱 구하고, 그 값을 출력하라.

AddMul.c

```
1 #include <stdio.h>
2 int main() {
3     int a, b;           // int형 변수 a와 b를 선언
4     int sum, mul;       // int형 변수 sum과 mul을 선언
5     scanf("%d %d", (&a), (&b)); // a와 b에 값을 입력
6     sum = a + b;
7     mul = a * b;
8     printf("%d + %d = %d", a, b, sum); // a + b = sum 출력
9     printf("%d * %d = %d", a, b, mul); // a * b = mul 출력
10 }
```

10 20 Enter

10 + 20 = 30

10 * 20 = 200

—

(5) 입력과 출력의 기초

■ [예제 2] 문자열을 입력하고, “Hello, 입력문자열” 형태로 출력하라.

- 문자열 입력을 위한 양식 변환기호: %s

Hello.c

```
1 #include <stdio.h>
2 int main() {
3     char str[100]; // 문자열을 입력할 문자 배열
4     printf("Who RU? ");
5     scanf("%s", str); // str에 문자열을 입력
6     printf("Hello, %s", str); // "Hello, 입력문자열!" 출력
7 }
```

```
Who RU? KNOU Enter
Hello, KNOU!
```

(4) 선행처리기

■ 선행처리(preprocessing)란?

- 컴파일에 앞서 소스 프로그램을 가공하여 컴파일러가 실제로 번역할 소스 프로그램을 만드는 것

```
1 #include <stdio.h>
2 int main() {
3     printf("Hello, C 프로그래밍\n");
4 }
```

(4) 선행처리기

■ 선행처리(preprocessing)란?

- 선행처리 지시어(preprocessing directives)로 지시함
 - 선행처리 지시어는 '#'로 시작함
 - 선행처리 지시어 문장은 한 행에 한 개의 문장을 작성함
 - 문장의 끝에 ';'을 사용하지 않음
- 대표적인 선행처리
 - 헤더파일 포함: #include
 - 매크로 선언 및 해제: #define, #undef
 - 조건부 컴파일: #if, #else, #elif, #endif

(4) 선행처리기

■ 헤더파일 포함: `#include` 지시어 사용

a.c

```
#include "a.h"  
문장_1;  
문장_2;  
문장_3;  
.....
```

a.h

```
문장_h1;  
문장_h2;  
문장_h3;
```

선행처리 결과

```
문장_h1;  
문장_h2;  
문장_h3;  
문장_1;  
문장_2;  
문장_3;  
.....
```

(4) 선행처리기

■ 헤더파일 포함: **#include** 지시어 사용

● 사용 형식

형식1

```
#include <파일명>
```

⇒ 표준 디렉토리에서 파일을 찾음

예

```
#include <stdio.h>
```

```
.....
```

stdio.h

stdlib.h

math.h

...

(4) 선행처리기

■ 헤더파일 포함: **#include** 지시어 사용

● 사용 형식

형식2

`#include "파일명"`

➔ 현재 사용 중인 디렉토리, 또는 직접 지정한 경로에서 파일을 찾음

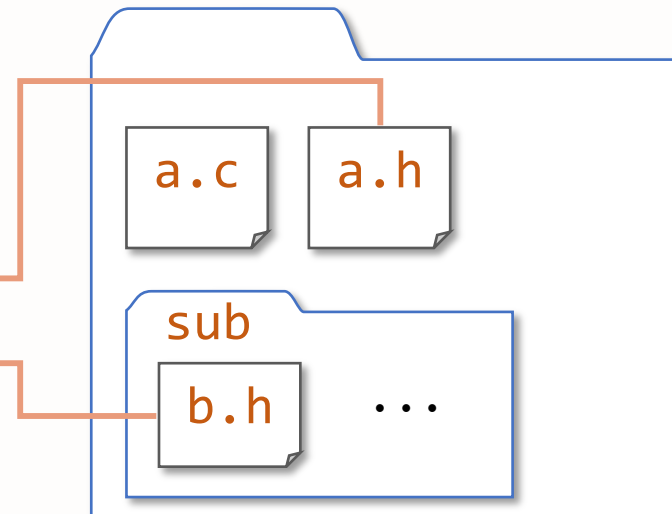
예

```
// a.c
```

```
#include "a.h" ←
```

```
#include "sub/b.h" ←
```

```
.....
```



(4) 선행처리기

■ 매크로 정의

- 매크로(macro)란?

- 특정 코드 패턴으로 치환되도록 정의된 명칭
- '**#define**'을 사용하여 자주 사용되는 명령이나 수식 또는 상수에 이를 대표하는 이름(매크로 이름)을 붙여 사용

- 매크로의 유형

- 매크로 상수
- 매크로 함수

(4) 선행처리기

■ 매크로 상수 정의

형식

```
#define 매크로명 값
```

→ 매크로 확장: 프로그램 내의 '매크로명'을 '값'으로 치환함

예

```
#define PI 3.141592
```

→ 이후에 나오는 'PI'는 '3.141592'로 치환함

■ 매크로 상수 해제

형식

```
#undef 매크로명
```


(4) 선행처리기

■ 매크로 상수 활용 예

```
1 #include <stdio.h>
2 #define PI 3.141592
3 int main() {
4     double r = 10.0;
5     printf("%lf\n", PI * r * r);
6 }
```



```
1 #include <stdio.h>
2 int main() {
3     double r = 10.0;
4     printf("%lf\n", 3.141592 * r * r);
5 }
```

(4) 선행처리기

■ 매크로 함수 정의

형식

```
#define 매크로명(인수리스트) (식)
```

- ▶ 인수리스트: 1개 이상의 인수
- ▶ 매크로 확장: 인수를 식에 반영하여 매크로명을 치환함
- ▶ **주의:** 매크로명이 (식)의 텍스트로 치환되는 것이므로 적절히 괄호를 사용할 필요가 있음

예

```
#define C_AREA(x) (3.141592 * (x) * (x))
```

(4) 선행처리기

■ 매크로 함수 활용 예

```
1 #include <stdio.h>
2 #define C_AREA(x) (3.141592 * (x) * (x))
3 int main() {
4     double r = 10.0;
5     printf("%lf\n", C_AREA(r));
6 }
```



```
1 #include <stdio.h>
2 int main() {
3     double r = 10.0;
4     printf("%lf\n", (3.141592 * (r) * (r)));
5 }
```

314.159200

(4) 선행처리기

■ 매크로 함수 활용 예

```
1 #include <stdio.h>
2 #define C_AREA(x) (3.141592 * (x) * (x))
3 int main() {
4     double r = 5.0;
5     printf("%lf\n", C_AREA(r + 5));
6 }
```



```
1 #include <stdio.h>
2 int main() {
3     double r = 5.0;
4     printf("%lf\n", (3.141592 * (r + 5) * (r + 5)));
5 }
```

314.159200

(4) 선행처리기

■ 매크로 함수 활용 예

```
1 #include <stdio.h>
2 #define C_AREA(x) (3.141592 * x * x)
3 int main() {
4     double r = 5.0;
5     printf("%lf\n", C_AREA(r + 5));
6 }
```



```
1 #include <stdio.h>
2 int main() {
3     double r = 5.0;
4     printf("%lf\n", (3.141592 * r + 5 * r + 5));
5 }
```



의도와 다른 매크로 확장

45.707960

(4) 선행처리기

- [예제 2] 다음 세 개의 소스코드에서 중복되는 부분을 헤더파일로 만들어 수정하라.

main.c

```
1 #include <stdio.h>
2 #define A 1
3 #define B 2
4 #define C 3
5 #define mul(x, y, z) ((x) * (y) * (z))
6 int f(int x1, int x2, int x3) {
7     return mul(x1, x2, x3);
8 }
9 int g(int x1, int x2, int x3) {
10     return mul(x1 + A, x2 + B, x3 + C);
11 }
12 }
```

f.c

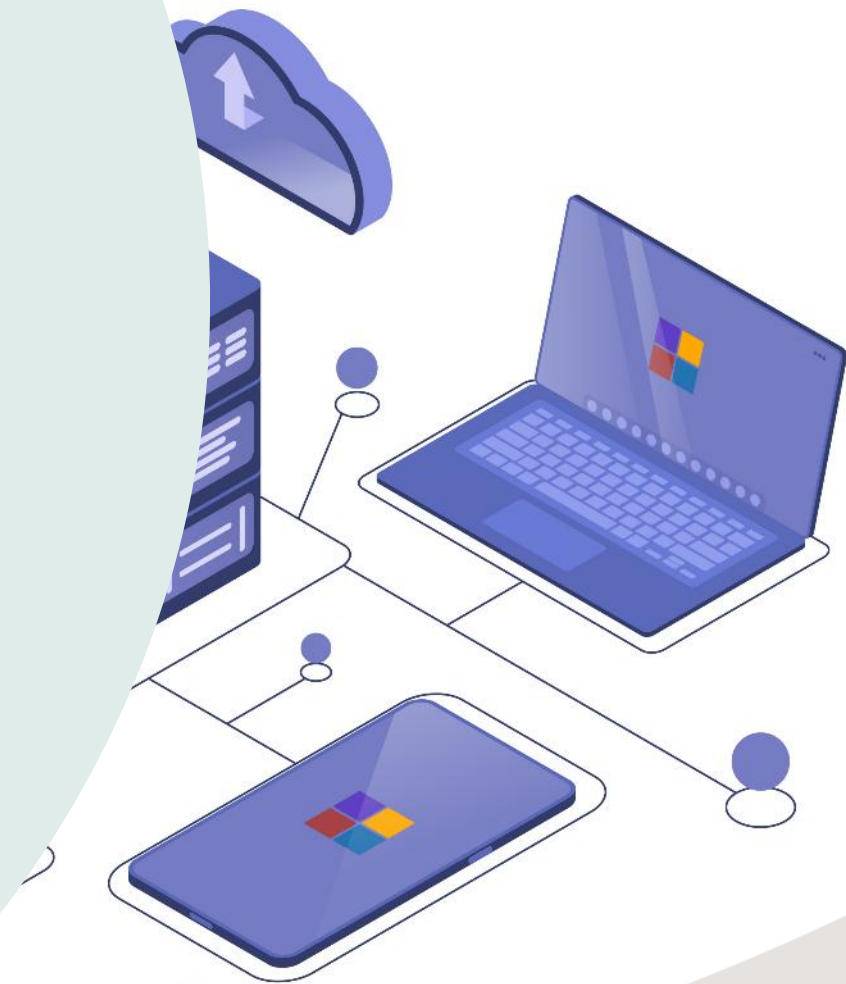
```
1 #define A 1
2 #define B 2
3 #define C 3
4 #define mul(x, y, z) ((x) * (y) * (z))
5
6 int f(int x1, int x2, int x3) {
7     return mul(x1, x2, x3);
8 }
```

g.c

```
1 #define A 1
2 #define B 2
3 #define C 3
4 #define mul(x, y, z) ((X) * (y) * (z))
5
6 int g(int x1, int x2, int x3) {
7     return mul(x1 + A, x2 + B, x3 + C);
8 }
```

C프로그래밍 출석수업

2 자료형과 상수 및 변수



(1) 자료형의 개념

■ 자료형(data type)이란?

- 컴퓨터에서 값의 표현 방법을 정의한 것
 - 값의 종류에 따른 표현 방법
 - ▶ 각 자료형에 해당되는 2진수 표현 방법을 사용함
 - 값을 표현하기 위한 메모리 공간
 - 값을 대상으로 수행할 수 있는 연산
- 값을 저장하거나 계산을 할 때 자료형을 엄격하게 구분하여 처리함
 - 필요한 경우 자료형을 변환하여 올바른 처리가 이루어질 수 있게 해야 함

(2) C 언어의 자료형

■ 기본형(primitive types)

정수 자료형

정수형 short int int
long int long long int
문자형 char

➡ signed 또는 unsigned

실수 자료형

float double
long double

■ 열거형(enumerated type)

■ 파생형(derived types)

- 배열형(array type), 구조체형(structure type), 공용체형(union type), 포인터형(pointer type)

(3) C 언어의 기본 자료형

■ 정수 자료형(integral types)

- 고정소수점(fixed-point) 방식의 숫자 표현
 - 오버플로(overflow)가 발생하지 않도록 주의해야 함
- signed형과 unsigned형으로 표현할 수 있음
 - signed가 디폴트임
- 값을 저장하기 위한 메모리의 크기
 - $\text{sizeof(char)} \leq \text{sizeof(short)} \leq \text{sizeof(int)} \leq \text{sizeof(long)} \leq \text{sizeof(long long)}$
- 메모리의 크기가 확정되어 있지는 않으며, C 언어를 구현하는 컴퓨터에서 가장 효율적으로 처리할 수 있는 정수형을 int형으로 표현함

(3) C 언어의 기본 자료형

■ 정수 자료형(integral types)

- signed 자료형 값의 범위(32 또는 64비트 컴퓨터 기준)

자료형	크기(byte)	값의 범위
char	1	-128~127
short int	2	-32,768~32,767
int	4	-2,147,483,648~2,147,483,647
long int	4	-2,147,483,648~2,147,483,647
long long int	8	-9,223,372,036,854,775,808~ 9,223,372,036,854,775,807

(3) C 언어의 기본 자료형

■ 정수 자료형(integral types)

- unsigned 자료형 값의 범위(32 또는 64비트 컴퓨터 기준)

자료형	크기(byte)	값의 범위
char	1	0~255
short int	2	0~65,535
int	4	0~4,294,967,295
long int	4	0~4,294,967,295
long long int	8	0~18,446,744,073,709,551,615

(2) C 언어의 기본 자료형

■ 정수 자료형(integral types)

- 표현할 수 있는 값의 범위는 `limits.h`에 정의된 값으로 확인할 수 있음
- 자료형의 저장공간 크기(바이트 단위)는 `sizeof` 연산자로 구할 수 있음

IntLimits.c

```
1 #include <stdio.h>
2 #include <limits.h>
3 int main() {
4     int minInt = INT_MIN;
5     int maxInt = INT_MAX;
6     printf("int의 크기: %zu\n", sizeof(int));
7     printf("int 자료형 범위: %d ~ %d\n", minInt, maxInt);
8 }
```

int의 크기: 4

int 자료형 범위: -2147483648 ~ 2147483647

—

(3) C 언어의 기본 자료형

■ 정수 자료형(integral types)

- [예제 4-1] short int형의 저장공간 크기와 표현할 수 있는 값의 범위 알아보기

ShortLimits.c

```
1 #include <stdio.h>
2 #include <limits.h>
3 int main() {
4     short int minShrt = ;
5     short int maxShrt = ;
6     printf("short int의 크기: %zu\n", );
7     printf("short int 자료형 범위: %d ~ %d\n", minShrt, maxShrt);
8 }
```

short int의 크기: 2

short int 자료형 범위: -32768 ~ 32767

—

(3) C 언어의 기본 자료형

■ 정수 자료형(integral types)

● 문자형의 사용 예

CharType.c

```
1 #include <stdio.h>
2 int main() {
3     char ch1;
4     ch1 = 'A';
5     printf("ch = '%c'\n", ch1);
6     printf("'%'의 ASCII 코드 = %d\n", ch1, ch1);
7     char ch2 = 0x42;
8     printf("ASCII 코드 %d의 문자 = '%c'\n", ch2, ch2);
9 }
```

ch = 'A'

'A'의 ASCII 코드 = 65

ASCII 코드 66에 해당되는 문자 = 'B'

—

(3) C 언어의 기본 자료형

■ 실수형

● 부동소수점 표현방식의 수(IEEE754 기준)

자료형	크기(byte)	값의 범위
float	4	$1.175494351 \times 10^{-38}$ $\sim 3.402823466 \times 10^{38}$
double	8	$2.2250738585072014 \times 10^{-308}$ $\sim 1.7976931348623158 \times 10^{308}$
long double	8	$2.2250738585072014 \times 10^{-308}$ $\sim 1.7976931348623158 \times 10^{308}$

(3) C 언어의 기본 자료형

■ 실수형

- 표현할 수 있는 값의 범위는 `float.h`에 정의된 값으로 확인할 수 있음

DoubleLimits.c

```
1 #include <stdio.h>
2 #include <float.h>
3 int main() {
4     double minDouble = DBL_MIN;
5     double maxDouble = DBL_MAX;
6     printf("double의 크기: %zu\n", sizeof(double));
7     printf("double 자료형 범위: %e ~ %e\n",
8         minDouble, maxDouble);
9 }
```

double의 크기: 4
double 자료형 범위: 2.225074e-308 ~ 1.797693e+308
—

(3) C 언어의 기본 자료형

■ 실수형

- [예제 4-2] float형의 저장공간 크기와 표현할 수 있는 값의 범위 알아보기

FloatLimits.c

```
1 #include <stdio.h>
2 #include <float.h>
3 int main() {
4     float minFlt = ;
5     float maxFlt = ;
6     printf("float의 크기: %zu\n", );
7     printf("float 자료형 범위: %e ~ %e\n",
8           minFlt, maxFlt);
9 }
```

float의 크기: 4

float 자료형 범위: 1.175494e-38 ~ 3.402823e+38

—

(4) 상수(constant)

■ 상수란?

- 항상 고정된 값을 갖는 자료

■ C 언어의 상수 표현

- 자료형에 따라 정해진 문법에 맞게 값을 표기함

예

'a'

123

1.2e-3

"KNOU"

(4) 상수(constant)

■ 정수형 상수

● 10진수, 8진수, 16진수로 표현

구 분	예	비 고
10진 상수	10, -45, 999	0~9 범위의 숫자를 사용함 숫자의 첫 자리는 0이 아니어야 함
8진 상수	012, -055	0~7 범위의 숫자를 사용함 숫자의 첫 자리는 0으로 시작함
16진 상수	0x0c, -0X2D	0~9, a~f를 사용함 숫자의 첫 자리는 0x로 시작함
unsigned형	10u, 012u, 0X0CU	u: 부호 없는 상수를 표현하는 접미사
long형	12345l, 0XFFL	l: long형의 상수를 표현하는 접미사

(4) 상수(constant)

■ 실수형 상수

- 부동소수점 형 상수
- double형이 기본 자료형임

구 분	예	비 고
소수 형식	3.14, -12.345	소수점을 사용하여 표현
지수 형식	1.2e3, 5e-2	10진수와 e를 사용하여 표현(1.2×10^3 , 5.0×10^{-2})
float형	3.14f, 0.314E1F	f: float형을 표현하는 접미사
long double형	3.14l, 0.314E1L	l: long double형을 표현하는 접미사

(4) 상수(constant)

■ 문자형 상수

- 작은따옴표(')로 묶여 있는 1개의 문자
 - 영문자: 'a' ~ 'z', 'A' ~ 'Z'
 - 숫자: '0' ~ '9'
 - 기호: '+', '-', '@', '#' 등
 - 백색문자(white space): 보이지는 않지만 토큰을 구분해 주는 역할을 하는 문자
 - ▶ ' ', '\t', '\n', '\r', '\f' 등
 - 널(null) 문자('\0')
- 내부적으로는 해당문자의 ASCII 코드값 저장

(4) 상수(constant)

■ 문자형 상수

- 특수한 문자의 표현: escape 문자 '\ '를 이용하여 표현
- 키보드에 나타나지 않는 문자

Escape 문자	기 능
' \a '	경고음(alarm) 출력
' \b '	백 스페이스(back space)
' \f '	새 페이지(form feed)
' \n '	줄 바꿈(new line)
' \r '	행의 시작 위치로 이동(carriage return)
' \t '	수평 탭(horizontal tab)
' \0 '	널(null) 문자(ASCII 코드값이 0인 문자)

(4) 상수(constant)

■ 문자형 상수

- 특수한 문자의 표현: escape 문자 '\ '를 이용하여 표현
- 기본적인 방법으로 표현할 수 없는 문자
 - ▶ 따옴표, '\ ' 문자 등

Escape 문자	기 능
'\ ' , '\ "'	작은따옴표, 큰따옴표
'\\ '	백슬래시('\')

- ASCII 코드로 문자를 표현하는 방법
 - ▶ '\8진수코드' 또는 '\x16진수코드'
 - 예: 문자 'A'의 표현 → '\101' 또는 '\x41'

(4) 상수(constant)

■ 문자열 상수

- 큰따옴표(" ")로 묶여 있는 일련의 문자
- 📝 문자열의 끝을 나타내기 위한 널 문자('\0')가 추가됨

예

문자열: "SEOUL KOREA"

⇒ S E O U L K O R E A \0

(5) 변수(variables)

■ 변수란?

- 프로그램에서 값을 저장하기 위한 기억공간

■ 변수의 선언

- 모든 변수는 사용하기 전에 미리 선언되어야 함
- 변수 선언 시 고려해야 할 사항

- ① 변수에 저장될 값의 크기
- ② 변수의 선언 위치
- ③ 변수의 초기화

(5) 변수(variables)

■ 변수 선언 시 고려해야 할 사항

- 오버플로(overflow), 언더플로(underflow)
- 표현할 수 있는 값의 범위를 벗어나는 문제

OverUnderflow.c

```
1 #include <stdio.h>
2 int main() {
3     short int num1 = 32767, num2 = -32768;
4     num1 = num1 + 1;
5     num2 = num2 - 1;
6     printf("num1 = %d\n", num1);
7     printf("num2 = %d\n", num2);
8 }
```

```
num1 = -32768
num2 = 32767
```

(5) 변수(variables)

■ 변수 선언 시 고려해야 할 사항

- 오버플로(overflow), 언더플로(underflow)
- 해결 방법: 값을 수용할 수 있는 자료형을 사용

OverUnderflow.c

```
1 #include <stdio.h>
2 int main() {
3     int num1 = 32767, num2 = -32768;
4     num1 = num1 + 1;
5     num2 = num2 - 1;
6     printf("num1 = %d\n", num1);
7     printf("num2 = %d\n", num2);
8 }
```

num1 = 32768

num2 = -32769

(5) 변수(variables)

■ 변수 선언 시 고려해야 할 사항

● 변수의 선언 위치

```
int num;           → 전역변수
int main() {
    int i;         → 지역변수
    .....
}
void sub() {
    int i;         → 지역변수 (main의 i와는 별개임)
    int j;
    .....
}
```

(5) 변수(variables)

■ 변수 선언 시 고려해야 할 사항

● 변수의 선언 위치

Scope.c

```
1 #include <stdio.h>
2 int a = 100;
3 void func() {
4     int a = 200;
5     printf("func()에서 a의 값 ==> %d\n", a);
6 }
7 int main() {
8     printf("main()에서 a의 값 ==> %d\n", a);
9     func();
10 }
```

main()에서 a의 값 ==> 100

func()에서 a의 값 ==> 200

—

(5) 변수(variables)

■ 변수 선언 시 고려해야 할 사항

- [예제 5] 다음 두 코드는 어떤 출력을 내는가?

InitL.c

```
1  #include <stdio.h>
2  int f() {
3      int x = 0;
4      x = x + 1;
5      return x;
6  }
7  int main() {
8      printf("%d\n", f());
9      printf("%d\n", f());
10     printf("%d\n", f());
11 }
```

InitG.c

```
1  #include <stdio.h>
2  int x = 0;
3  int f() {
4      x = x + 1;
5      return x;
6  }
7  int main() {
8      printf("%d\n", f());
9      printf("%d\n", f());
10     printf("%d\n", f());
11 }
```

(5) 변수(variables)

■ 변수 선언 시 고려해야 할 사항

● 변수의 초기값

- 변수의 값을 사용하기 전에 적절한 값이 저장되어 있어야 함
- 지역변수의 초기값

▶ 초기화를 생략하면 예측할 수 없는 값(garbage value)을 가짐

• 예: `int a;` // a의 값을 예측할 수 없음

▶ 변수를 선언할 때 초기값을 제공하여 초기화를 할 수 있음

• 예: `int a = 10;` // 초기값으로 0이 지정된 a를 선언함

● 전역변수의 초기값

▶ 초기화를 생략하면 초기값으로 0이 지정됨

▶ 필요한 경우 변수 선언문에 초기값을 제공할 수 있음

(5) 변수(variables)

■ 변수 선언 시 고려해야 할 사항

● 변수의 초기값

InitialVaule.c

```
1 #include <stdio.h>
2 int a;
3 int b = 10;
4 int main() {
5     int c;
6     int d = 20;
7     printf("전역변수 a = %d, b = %d\n", a, b);
8     printf("지역변수 c = %d, d = %d\n", c, d);
9 }
```

전역변수 a = 0, b = 10

지역변수 c = 16, d = 20

(5) 변수(variables)

■ 변수 선언 시 고려해야 할 사항

● 변수의 초기값

Initialization.c

```
1 #include <stdio.h>
2 int main() {
3     int i, sum;
4     for (i = 1; i <= 10; i++)
5         sum = sum + i;
6     printf("1부터 10까지의 합 = %d\n", sum);
7 }
```



컴파일러 구현에 따라서는 초기화가 안 된 변수의 값을 사용할 경우 에러로 취급하기도 함

1부터 10까지의 합 = 32819

(5) 변수(variables)

■ 변수 선언 시 고려해야 할 사항

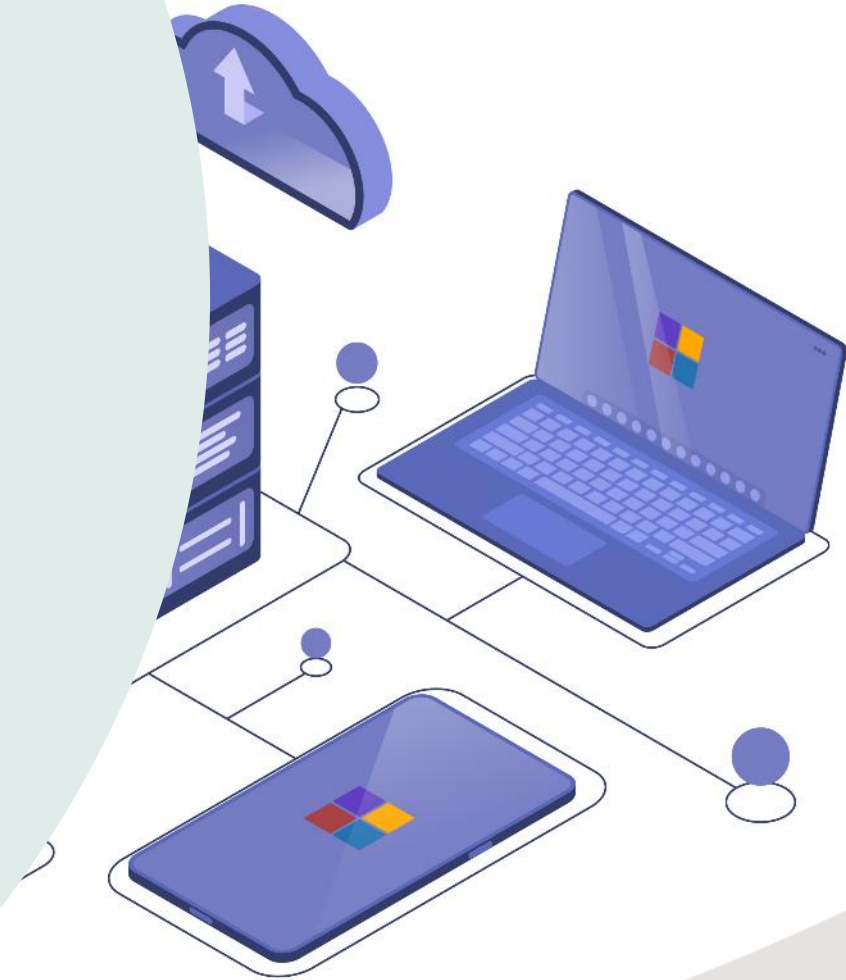
● 변수의 초기값

Initialization.c

```
1 #include <stdio.h>
2 int main() {
3     int i, sum = 0;
4     for (i = 1; i <= 10; i++)
5         sum = sum + i;
6     printf("1부터 10까지의 합 = %d\n", sum);
7 }
```

1부터 10까지의 합 = 55

3 연산자



(1) 연산자의 개념

■ 연산자(operator)란?

- 자료를 대상으로 각종 연산을 수행을 지시하는 기호

■ 연산자의 종류




구 분	연 산 자
산술 연산자	+ - * / % ++ --
관계 연산자	> < >= <= == !=
논리 연산자	&& !
대입 연산자	= += -= *= /= %= <<= >>= &= = ^=
조건 연산자	?:
비트 연산자	& ^ ~ << >>
기타 연산자	sizeof() , & * 형변환(type cast)

(2) 산술 연산자

■ 2항 연산자

● 사칙 연산자: $+$, $-$, $*$, $/$

수 식	수식의 값	비 고
$5 + 3$		정수형 덧셈
$8.0 / 5.0$		실수형 나눗셈
$8 / 5$		정수형 나눗셈

-  오버플로가 일어나지 않도록 주의해야 함
-  나눗셈의 제수는 0이 아니어야 함
-  정수형 나눗셈의 결과는 정수형: 소수점 이하 버림

(2) 산술 연산자

■ 2항 연산자

● 나머지 연산자: %

수식	수식의 값
5 % 3	
-5 % 3	
5 % -3	
-5 % -3	

 정수형에만 사용할 수 있음

 나머지의 정의: 피제수 - (피제수 / 제수) * 제수

(2) 산술 연산자

■ 단항 연산자

- 증, 감 연산자: ++, --

수식(a가 5일 때)	실행 결과	
	a의 값	b의 값
b = ++a;		
b = a++;		
b = --a;		
b = a--;		

- 📌 전치 연산식의 값: 변화된 값
- 📌 후치 연산식의 값: 변화되기 전의 값

(3) 관계 연산자

■ 관계 연산자란?

- 두 피연산자에 대한 등호 및 부등호 연산자

연산자	의미	사용 예
==	같음	a == b
!=	다름	a != b
> >= < <=	대, 소 관계 비교	a >= b

(4) 논리 연산자

■ 논리 연산자란?

- 불 값을 대상으로 참·거짓을 구하는 연산자

연산자	기 능	사 용 예
&&	논리곱(AND)	a && b
	논리합(OR)	a b
!	부정(NOT)	!a

(5) 조건 연산자

■ 조건 연산자란?

- 주어진 조건이 참일 때와 거짓일 때의 값을 선택할 수 있는 연산자

형식

$expr_1 \text{ ? } expr_2 \text{ : } expr_3$

- ▶ $expr_1$: 조건식
- ▶ $expr_2$: 조건이 참일 때의 값을 구하는 식
- ▶ $expr_3$: 조건이 거짓일 때의 값을 구하는 식

- 예:

```
int a = 10, b = 20, max;  
max = a > b ? a : b;
```

(6) 대입 연산자

■ 대입 연산자란?

- 좌측 피연산자에 우측 피연산자의 값을 저장하는 연산자
- 대입 연산식의 값: 대입된 값

형식

$expr_l = expr_r;$

l-value r-value
(저장공간) (값)

- 예:

```
int a, b;  
b = 5;  
a = b + 10;  
a = b = 0;
```

(6) 대입 연산자

■ 복합 대입 연산자란?

- 2항 연산자의 좌측 피연산자에 연산 결과를 대입하는 형태의 대입 연산자

형식

$expr_1 \text{ } op = expr_2;$

▶ op : + - * / % << >> & ^ |

 ' $expr_1 = expr_1 \text{ } op \text{ } expr_2;$ '라는 표현과 동일함

● 예: $a \ += \ b; \quad // \ a \ = \ a \ + \ b;$

(7) 비트 연산자

■ 비트 연산자란?

- 정수형(문자형 포함) 값의 비트(bit) 단위로 연산을 수행하는 연산자

연산자	기능
a & b	a와 b의 대응되는 두 비트가 모두 1일 때 결과가 1
a b	a와 b의 대응되는 두 비트 중 하나라도 1이면 결과가 1
a ^ b	a와 b의 대응되는 두 비트가 서로 다를 때만 결과가 1
~a	각 비트의 1은 0으로, 0은 1로 바꾼 값
a << n	a의 각 비트를 n 비트씩 왼쪽으로 이동
a >> n	a의 각 비트를 n 비트씩 오른쪽으로 이동

(8) 기타 연산자

연산자	기 능
sizeof()	자료형이나 식(expression)이 차지하는 기억공간의 크기(byte)를 구함(size_t)
형변환	식의 자료형을 다른 자료형으로 강제로 바꿈
,	두 개의 식을 하나로 묶음
&	피연산자의 주소를 구함
*	포인터가 가리키는 곳 또는 그 곳의 내용

(8) 기타 연산자

■ 형변환(type cast)

- 식의 자료형을 다른 자료형으로 바꾸는 것
- 형변환 연산자: 명시적으로 형변환을 지시하는 연산자

형식

(typename)expr

- ▶ *typename*: 변환할 자료형
- ▶ *expr*: 자료형을 변환할 식
- ▶ 결과: *expr*을 *typename*형으로 변환한 값

- 예:

```
float a = 1.6;  
int b = (int)a;
```


(8) 기타 연산자

■ 형변환(type cast)

● 명시적 형변환의 예

```
1 #include <stdio.h>
2 int main() {
3     int a = 3, b = 4;
4     double c;
5     c = a / b;
6     printf("나눗셈 결과: %f\n", c);
7 }
```

나눗셈 결과: 0.000000

(8) 기타 연산자

■ 형변환(type cast)

● 명시적 형변환의 예

```
1 #include <stdio.h>
2 int main() {
3     int a = 3, b = 4;
4     double c;
5     c = (double) a / b;
6     printf("나눗셈 결과: %f\n", c);
7 }
```

나눗셈 결과: 0.750000

(8) 기타 연산자

■ 형변환(type cast)

● 묵시적 형변환

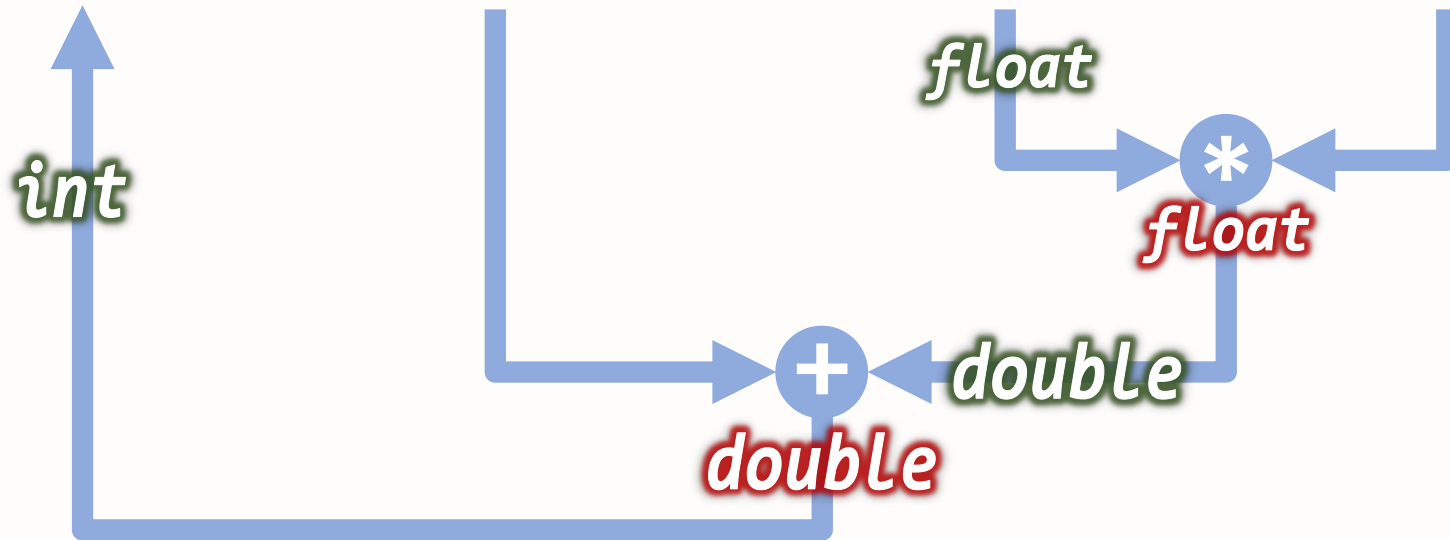
- 특별히 형변환 연산자를 사용하지 않아도 연산이 가능한 자료형으로 자동적으로 이루어지는 형변환
- 묵시적 형변환의 형태
 - ▶ 정수형 승격(integral promotion)
 - char, unsigned char, short, unsigned short 등의 int형보다 작은 자료형은 효율적인 처리를 위해 int형 또는 unsigned int형으로 변환됨
 - ▶ 서로 다른 자료형이 혼재하는 식
 - 표현 범위가 더 큰 자료형으로 변환됨
 - 연산 순서에 따라 꼭 필요한 지점에서 변환됨

(8) 기타 연산자

■ 형변환(type cast)

● 묵시적 형변환의 예

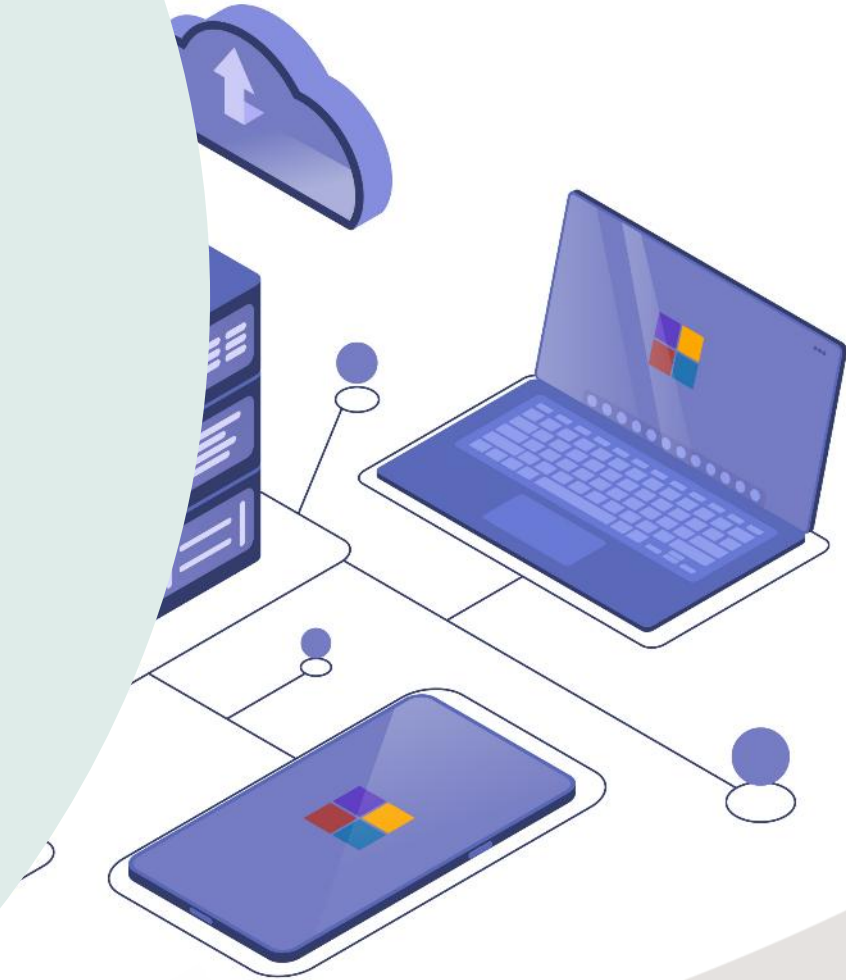
```
intVar = doubleVar + intVar * floatVar;
```



(9) 연산자의 우선순위와 결합방향

연산자명		연산자	결합방향	우선순위
괄호, 구조체, 공용체 연산자		() [] -> .	좌→우	<div>높음</div> <div>↑</div> <div>↓</div> <div>낮음</div>
단항 연산자		! ~ ++ -- & * sizeof() 형변환	우→좌	
이항 연산자	승, 제	* / %	좌→우	
	가, 감	+ -	좌→우	
	비트 이동	<< >>	좌→우	
	대소 비교	< <= > >=	좌→우	
	등가 판정	== !=	좌→우	
	비트 AND	&	좌→우	
	비트 XOR	^	좌→우	
	비트 OR		좌→우	
	논리 AND	&&	좌→우	
	논리 OR		좌→우	
조건 연산자		? :	우→좌	
대입 연산자		= += -= *= 등	우→좌	

4 흐름 제어



(1) C 언어의 기본적인 흐름 제어 구조

```
#include <stdio.h>
..... // 선언문
int main() {
```

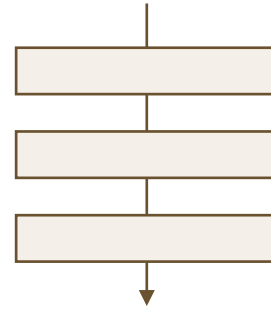
statements

```
.....
}

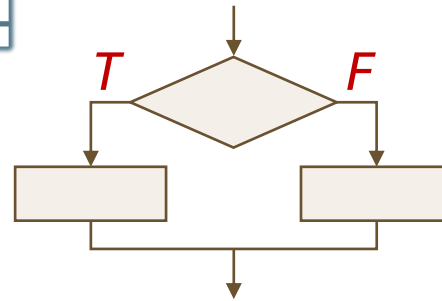
void f() {
.....
}

double g() {
.....
}
```

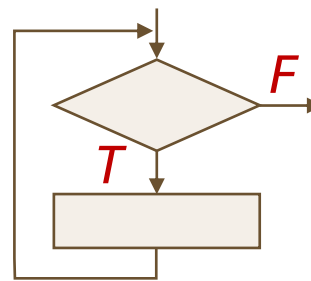
순차



선택



반복



- if 문
- switch 문
- for 문
- while 문
- do - while 문

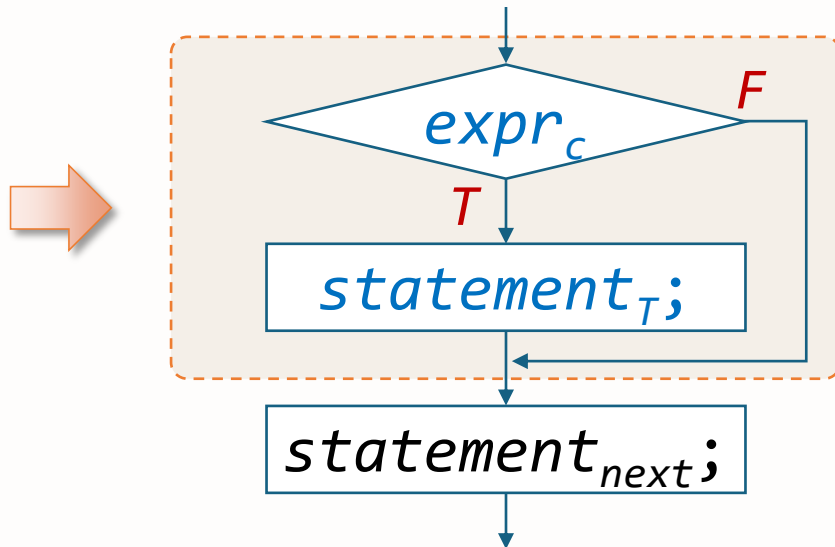
(2) 선택 제어문

■ 단순 if 문

형식

```

if (exprc)           // exprc : 조건
    statementT;       // exprc가 참이면 실행
statementnext;       // if 문에 이어서 실행
  
```



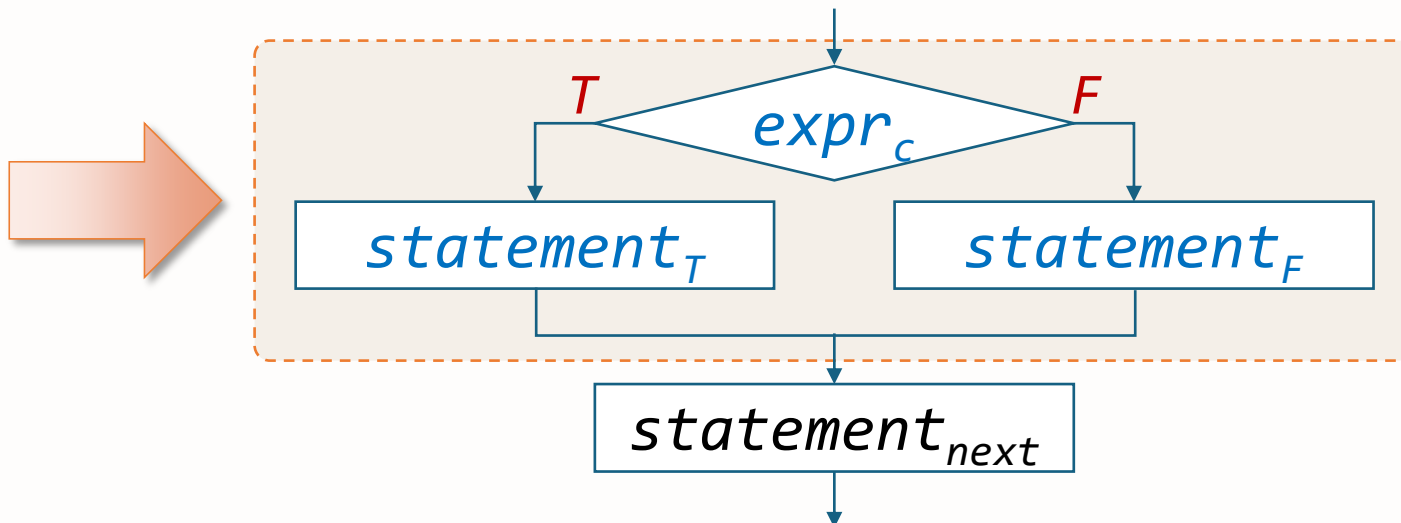
$statement_T$ 에 다수의 문장이
필요한 경우 **복합문**으로 작성함

(2) 선택 제어문

■ if ~ else 문

형식

```
if ( $expr_c$ )           //  $expr_c$  : 조건  
     $statement_T$ ;      //  $expr_c$ 가 참이면 실행  
else  
     $statement_F$ ;      //  $expr_c$ 가 거짓이면 실행  
 $statement_{next}$ ;    // if 문에 이어서 실행
```

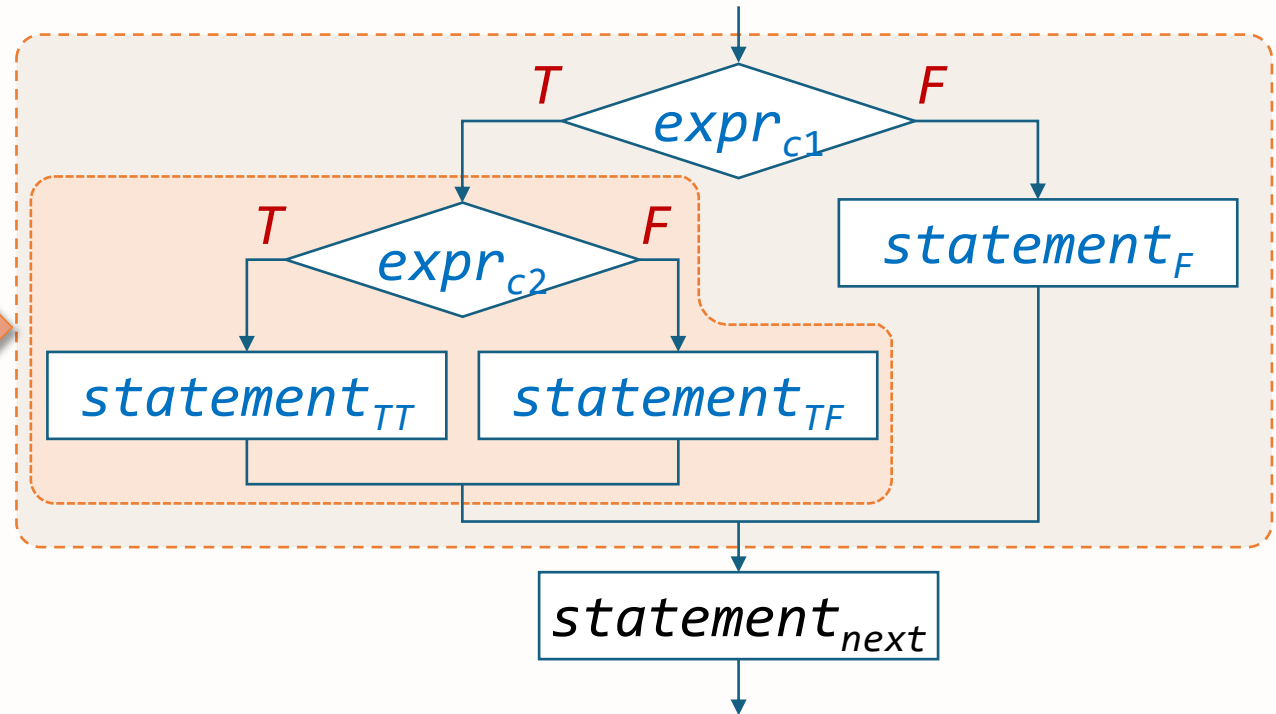


(2) 선택 제어문

■ 다중 if 문

형식

```
if (exprc1)  
    if (exprc2)  
        statementTT;  
    else  
        statementTF;  
else  
    statementF;  
statementnext;
```

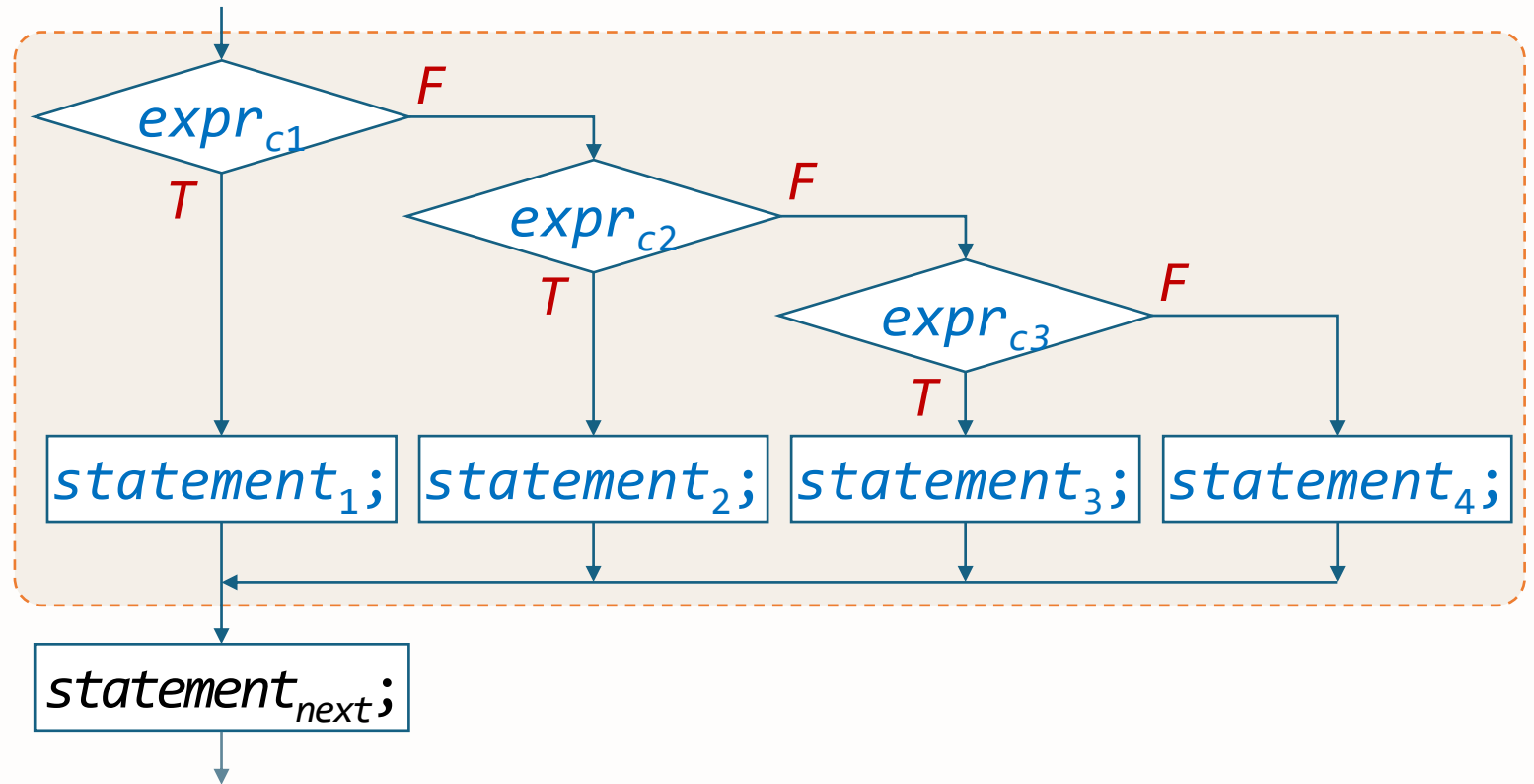


(2) 선택 제어문

다중 if ~ else if ~ else 문

형식

```
if (exprc1)  
    statement1;  
else if (exprc2)  
    statement2;  
else if (exprc3)  
    statement3;  
else  
    statement4;  
statementnext;
```



(2) 선택 제어문

■ switch 문

- 지정된 식의 값에 따라 실행할 문장을 선택할 수 있게 함

형식

```
switch ( $expr_{int}$ ) {  
  case  $label_1$ :  
    statements1;  
  case  $label_2$ :  
    statements2;  
  case  $label_3$ :  
    statements3;  
  default:  
    statementsd;  
}
```

 $expr_{int}$: 정수 유형의 식

▶ char, short, int,
long, long long,
열거형

 $label_i$: $expr_{int}$ 의
자료형으로 변환할 수
있는 상수

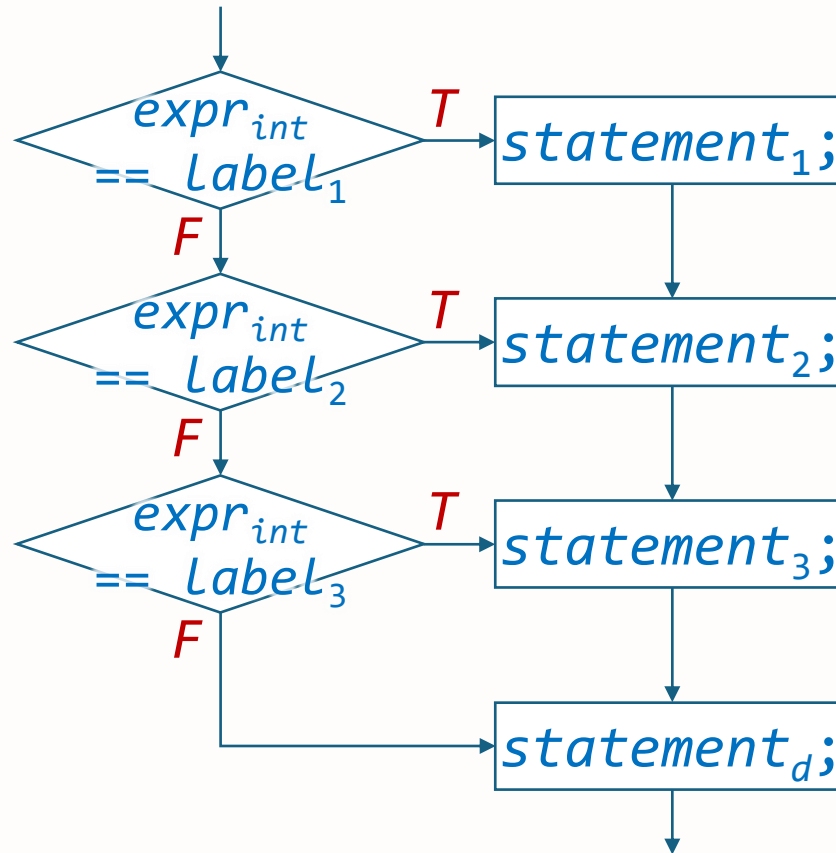
(2) 선택 제어문

■ switch 문

- 지정된 식의 값에 따라 실행할 문장을 선택할 수 있게 함

형식

```
switch ( $expr_{int}$ ) {  
  case  $label_1$ :  
    statements1;  
  case  $label_2$ :  
    statements2;  
  case  $label_3$ :  
    statements3;  
  default:  
    statementsd;  
}
```



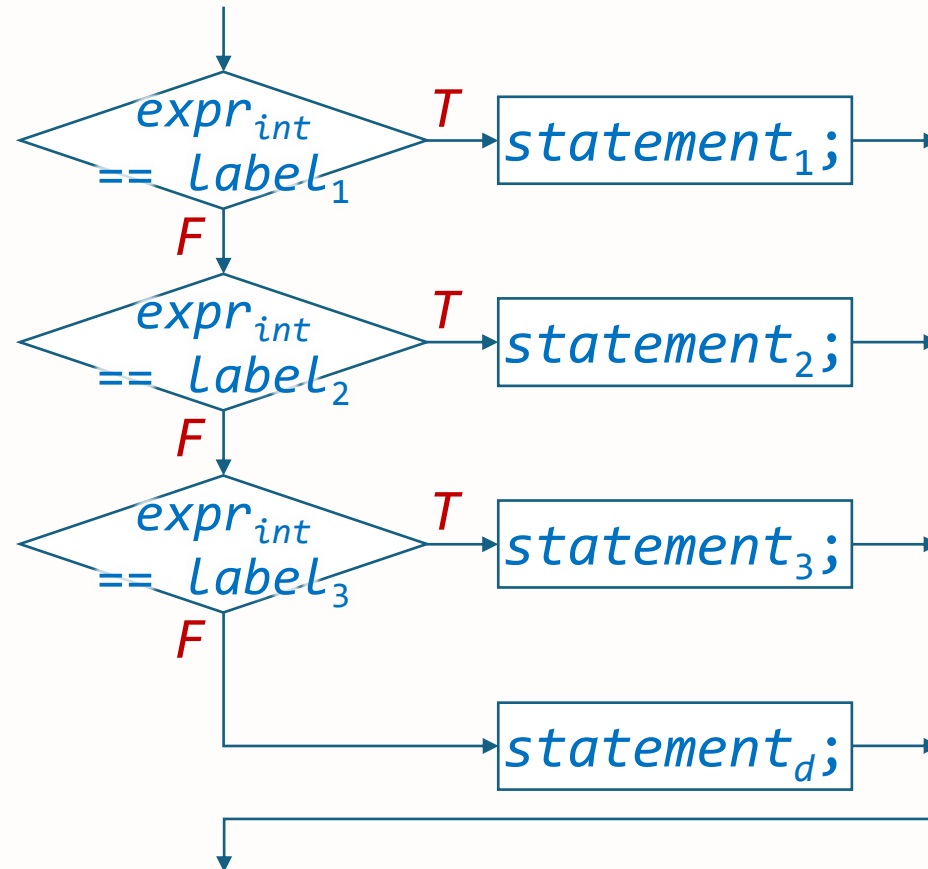
(2) 선택 제어문

■ switch 문

- 지정된 식의 값에 따라 실행할 문장을 선택할 수 있게 함

형식

```
switch ( $expr_{int}$ ) {
  case  $label_1$ :
    statements1;
    break;
  case  $label_2$ :
    statements2;
    break;
  case  $label_3$ :
    statements3;
    break;
  default:
    statementsd;
}
```

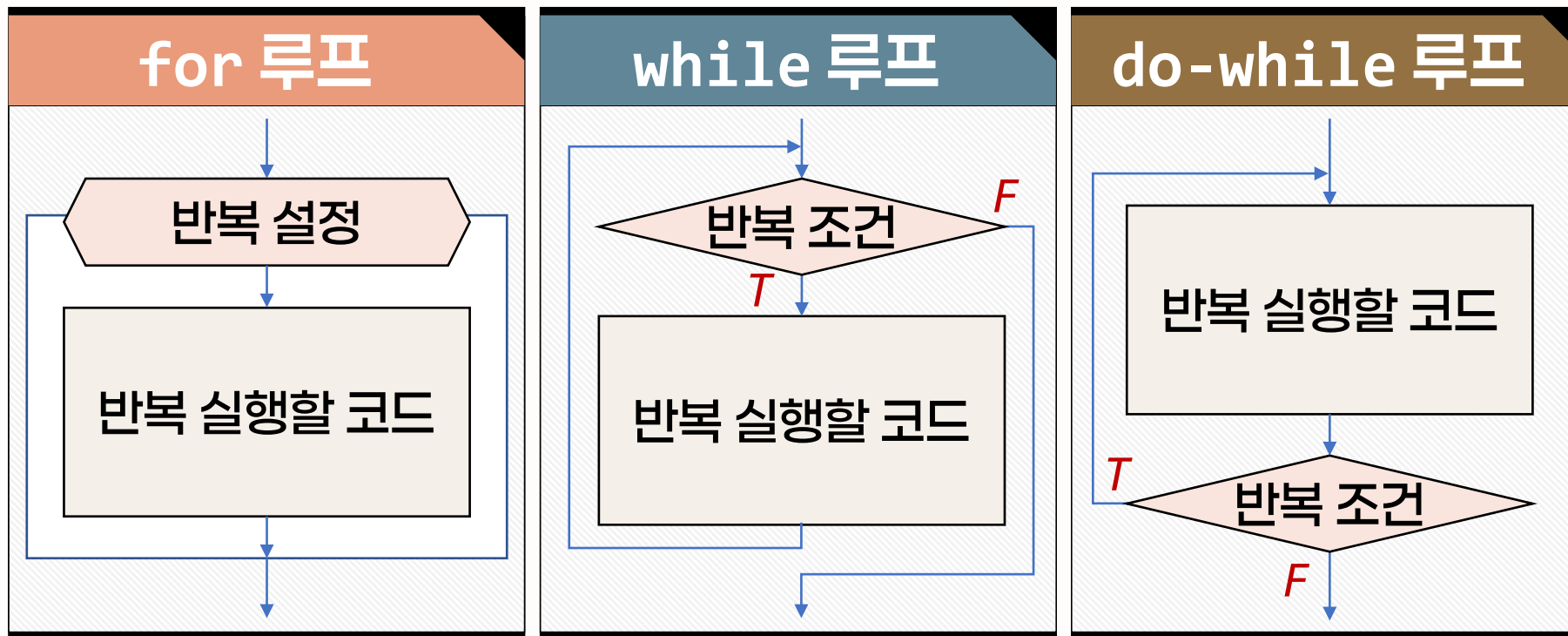


(3) 반복 제어문

■ 반복 제어문(loop, 루프)의 용도

- 정해진 조건에 따라 동일한 처리를 반복적으로 실행함

■ 반복 제어문의 유형



(3) 반복 제어문

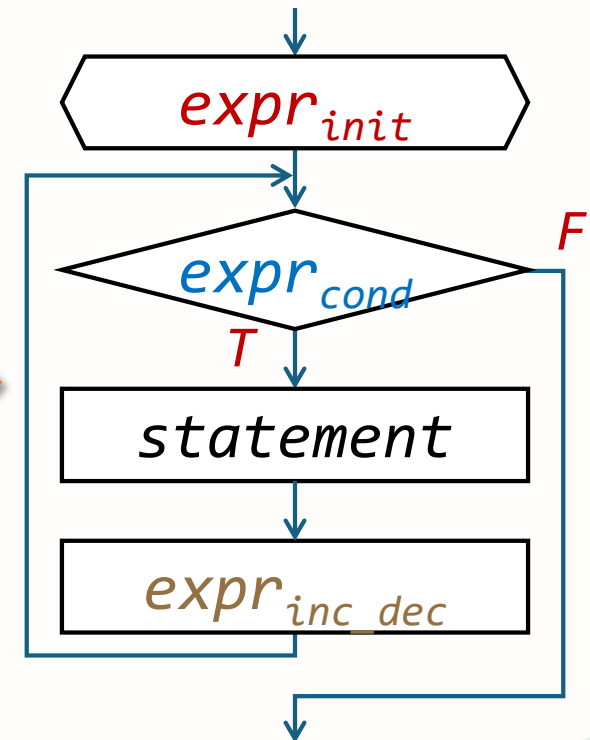
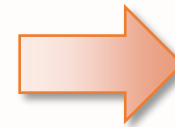
■ for 문

- 반복 횟수가 명확한 경우 주로 사용됨
- 반복 횟수를 카운트하기 위한 변수를 사용함

형식

```
for (exprinit; exprcond; exprinc_dec)  
    statement;
```

- ▶ *expr_{init}*: 반복 횟수 카운트 변수 초기화
- ▶ *expr_{cond}*: 반복을 하게 되는 조건
- ▶ *expr_{inc_dec}*: 카운트 변수를 증·감
- ▶ *statement*: 반복할 문장(또는 블록)



(3) 반복 제어문

■ for 문

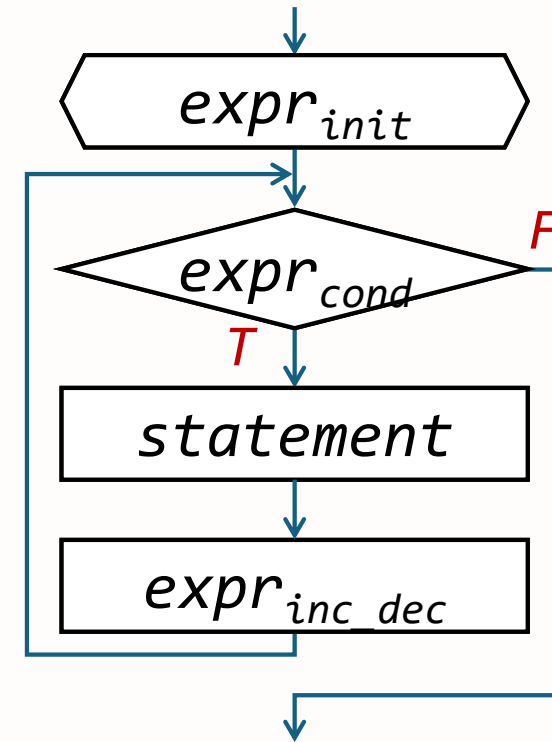
● 사용 예

```

1 #include <stdio.h>
2 int main() {
3     int i, sum = 0;
4     for (i = 1 ; i <= 10 ; ++i)
5         sum = sum + i;
6     printf("1부터 %d까지의 합 = %d\n",
7           i - 1, sum);
8 }

```

1부터 10까지의 합 = 55



(3) 반복 제어문

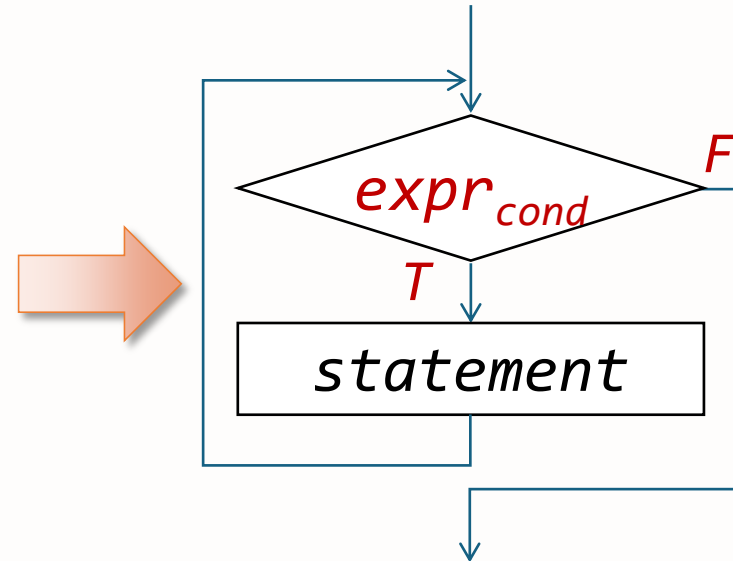
■ while 문

- 지정된 조건이 만족되는 동안 반복
- 초기의 조건이 거짓이면 반복문의 문장을 한 번도 실행하지 않을 수도 있음

형식

```
while (exprcond)
    statement;
```

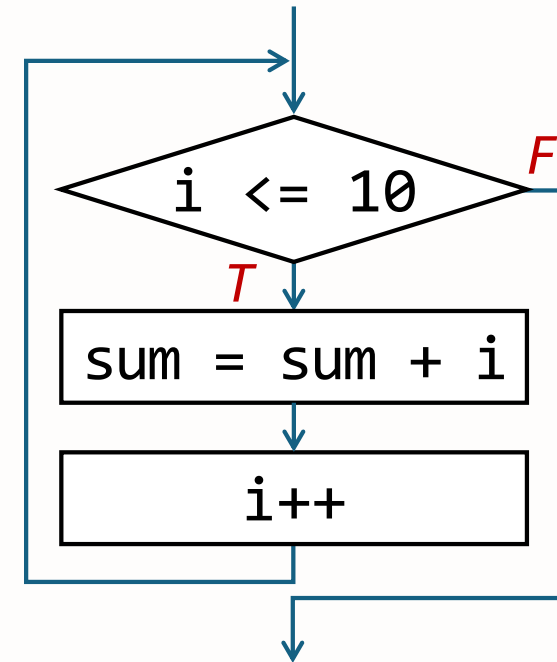
- ▶ *expr_{cond}*: 반복을 하게 되는 조건
- ▶ *statement*: 반복할 문장(또는 블록)



(3) 반복 제어문

■ while 문

```
1 #include <stdio.h>
2 int main() {
3     int i = 1;
4     int sum = 0;
5     while (i <= 10) {
6         sum = sum + i;
7         i++;
8     }
9     printf("1부터 %d까지의 합 = %d\n",
10         i - 1, sum);
11 }
```



(3) 반복 제어문

■ do - while 문

- 지정된 조건이 만족되는 동안 반복

📝 반복문 내의 문장을 실행한 후 반복 실행 조건을 검사함

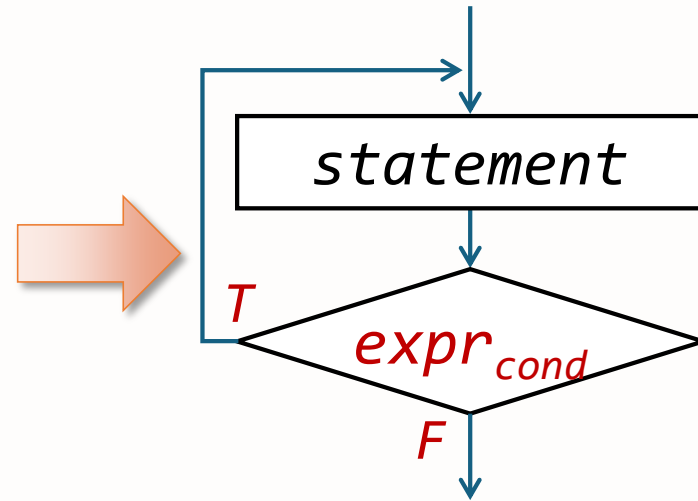
▶ 반복 실행할 문장을 최소 1회는 실행하게 됨

형식

```
do {
    statement;
} while (exprcond)
```

▶ *expr_{cond}*: 반복을 하게 되는 조건

▶ *statement*: 반복할 문장



(3) 반복 제어문

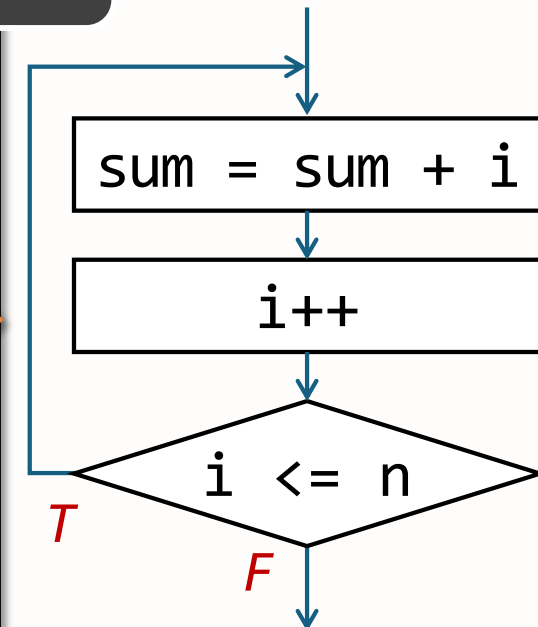
■ do - while 문

```
1 #include <stdio.h>
2 int main() {
3     int i = 1, n;
4     int sum = 0;
5     printf("n = ");
6     scanf("%d", &n);
7     do {
8         sum = sum + i;
9         i++;
10    } while (i <= n);
11    printf("i = %d\n", i);
12    printf("1부터 %d까지의 합 = %d\n", i - 1, sum);
13 }
```

n = 10 Enter

i = 11

1부터 10까지의 합 = 55



C프로그래밍

수고하셨습니다!

