is it clear : ||\

2510 : Lecture 17 || Feb. 27 / 17

▷ goto typical use :
↳ put error-handling code at the end of a function.
When there's an error, goto that specific section of code.

```
void f(...) {
  ...
  if(...)
    goto error;
  ...
error:
}
```

▷ ternary operator

this is expression NOT statement

eg.

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

▷ examples on SIMPLIFYING CODE

```
int is_valid_score(int n){
  if (0<=n && n<=100)
    return 1;
  else
    return 0;
}
```

↓

```
int is_valid_score(int n){
  if (0<=n && n<=100)
    return 1;
  return 0;
}
```

↓

```
int is_valid_score(int n){
  return 0<=n && n<=100 ? 1 : 0;
}
```

SIMPLEST

```
int is_valid_score(int n){
  return 0<=n && n<=100;
}
```

this already returns true or false

▷ Precedence + Associativity

↳ determines how we group terms

this is the correct version:
* has higher precedence

REFER to Albert's Precedence Table

`a + b * c` Is this the same as `(a + b) * c` or `a + (b * c)`?

`c = getchar() != EOF` `!=` has HIGHER precedence than `=`

therefore ↳

`c = getchar() != EOF` ≡ `c = (getchar() != EOF)`

`a + b + c`

look @ associative → `a - b + c` + and - have same precedence but associate LEFT to RIGHT

∴ `a - b + c` ≡ `(a - b) + c`

associate! `*p++` Is this: `*(p++)` or `(*p)++`?

but associate RIGHT to LEFT

∴ `*p++` ≡ `*(p++)`

▷ Order of Evaluation

In many cases, C does not specify the order of evaluation:

eg.
```
void f(int m, int n);
int g(int);
int h(int);
f(g(1), h(2));
```

the order of evaluations of the arguments to a function is unspecified

Is g or h called first??

↳ THIS IS NOT SPECIFIED BY C LIBRARY

would give different answers based on compilers

```
int n = 1;
f(++n, n++);
```

value of this expression is unspecified

DANGEROUS? ok...

In general, you shouldn't try to modify the same variable more than once in an expression: `int n = ++n + n++;`

... Lecture 17

Cont...

Some cases where the order of evaluations is specified:

&&, ||

- Examples

1) int a = 1, b = 0, c = 2;
c = --a && ++b; → not evaluated FALSE AND
printf("%d %d %d \n", a, b, c);
0 0 0

2) int a = 1, b = 0, c = 2;
c = a-- && ++b;
1 1 1
0 1
printf("%d %d %d \n" a, b, c);
0 1 1

3) int a = 1, b = 0, c = 2;
c = --a || ++b;
0 1
printf("%d %d %d \n", a, b, c);
0 1 1

4) int a = 1, b = 0, c = 2;
c = a-- || ++b; → not evaluated b/c already true.
printf("%d %d %d \n", a, b, c);
0 0 1

final material