Practice information Processing

(IMACU)

Third lecture (2nd part)
Selection and Iteration Process

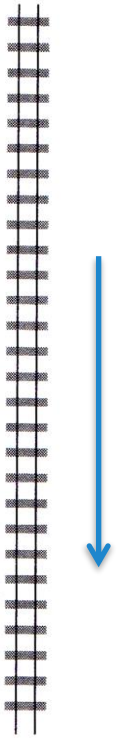
Makoto Hirota

Contents of the second part of this lecture

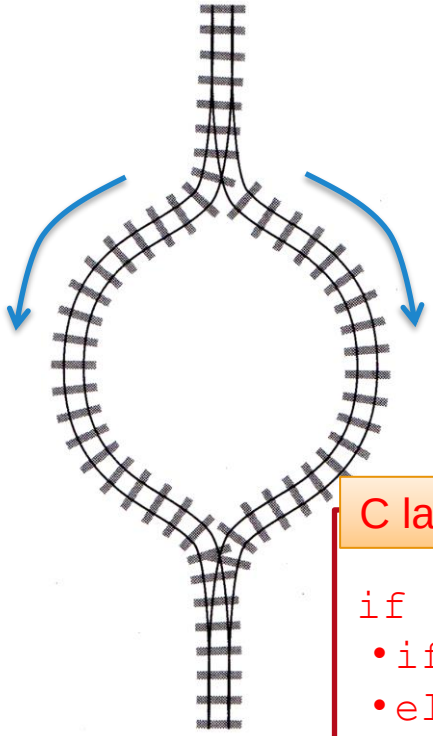
- Exapmple answer of previous exercise (`for` statement)
- PAD expression
- Iteration process (repeating) format No. 2
  - `while` statement
- Selection process (branching) format
  - `if` statement
    - `if`
    - `if else`
    - `else`
  - `switch` statement

Basic structure of processing flow

- There are only three basic forms of "processing flow" in a program



Sequence

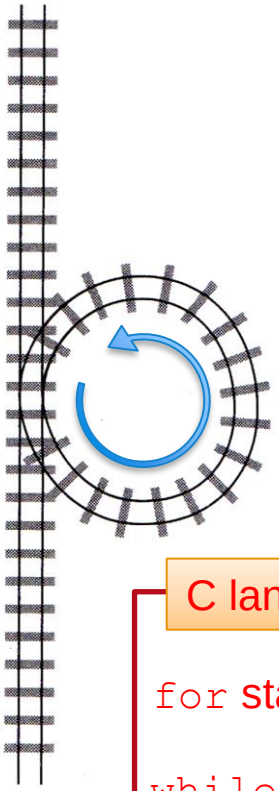


Selection

C language

`if` statement
- `if`
- `else if`
- `else`

`switch` statement



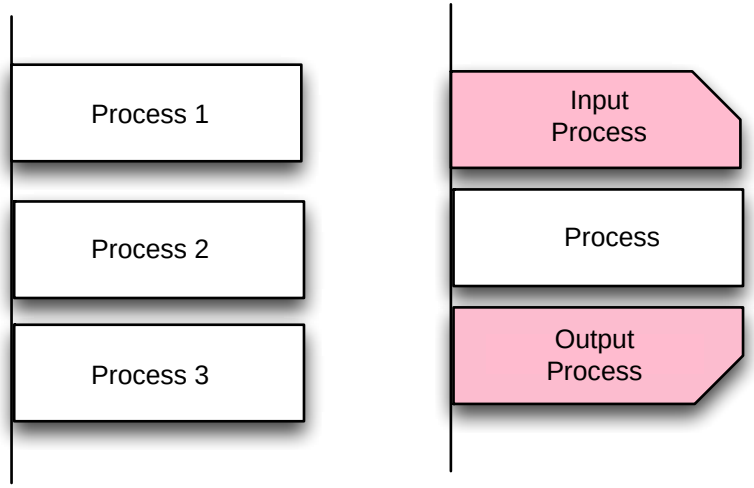
Iteration

C language

`for` statement

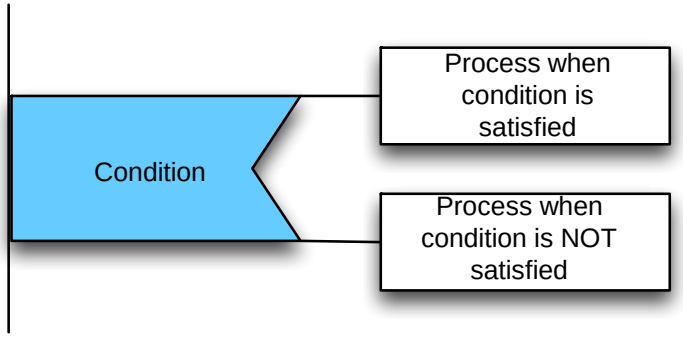
`while` statement (do-while)

Expression using PAD（Problem Analysis Diagram）

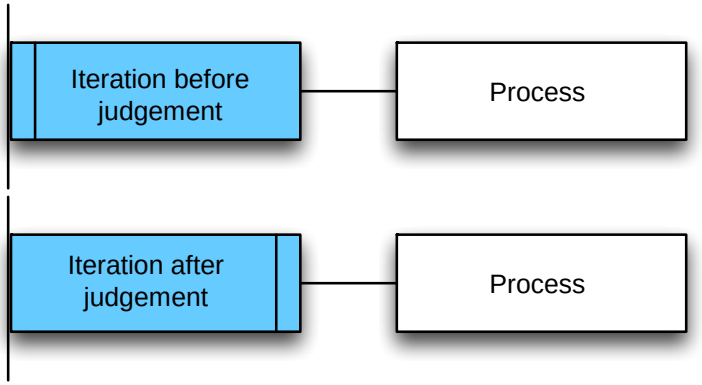


Sequence

Process in order of
Top to bottom
Left to right



Selection



Iteration

Kind of conditional statement

- Relational operator

Operator	Meaning
a < b	a ＜ b （a is smaller than b）
a <= b	a ≦ b （a is smaller than or equal to b）
a > b	a ＞ b （a is greater than b）
a >= b	a ≧ b （a is greater than or equal to b）
a == b	a ＝ b （a is equal to b）
a != b	a ≠ b （a is not equal to b）

Operators include “=“

<=
>=
!=

Remember ‘=’ is located later

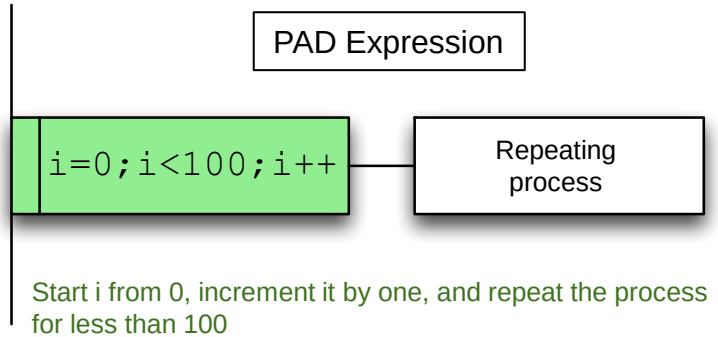
- Logical operator（Multiple conditions）

Operator	Meaning
Condition A && B	Logical product (AND) When condition A and condition B are satisfied
Condition A || B	Logical sum (OR) When condition A or condition B is satisfied
!Condition A	Negation (NOT) When condition A is not met

Iterative process

`for` (iteration of a certain number of times)

```
for (initial condition; continuous condition; incremental process)
{
    Repeating process ;
}
```

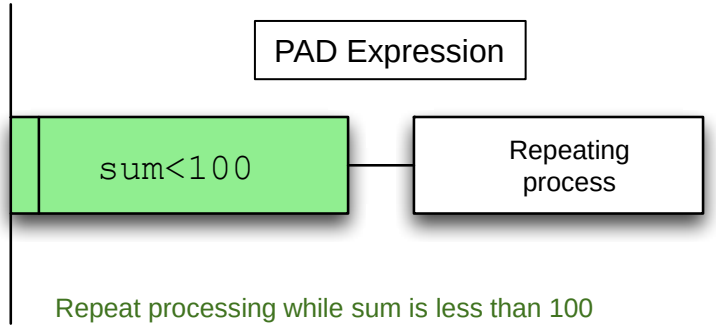


Start i from 0, increment it by one, and repeat the process for less than 100

`while` (Repeating while the condition is satisfied)

＝conditional expression is true

```
while (conditional expression)
{
    Repeating process ;
}
```



Repeat processing while sum is less than 100

Two ways escaping from while loop

1. Make the conditional expression false in the iterative process
2. Insert a `break` statement when a specific condition becomes true in iterative processing

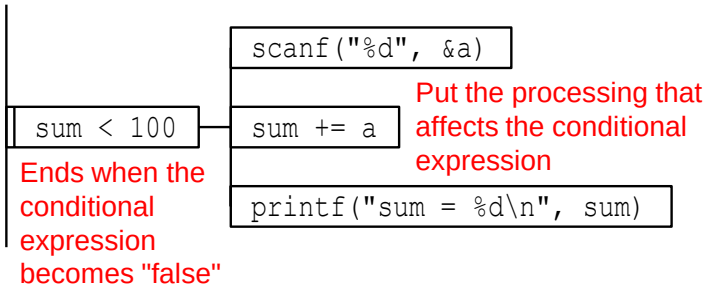
Ex: Escape from while loop： false condition

- Sample sum_up01.c

```c
#include <stdio.h>

int main()
    /**** variable declaration ****/
    int a;
    int sum = 0;
    /**** processing contents****/

    while(sum < 100){ /* while statement*/

        scanf("%d",&a); /* formated input */

        sum += a;

        printf("sum = %d¥n", sum);
    }

    printf("sum is more than 100¥n");

    return 0;
}
```

```
$ gcc -Wall -o sum_up01 sum_up01.c
$ ./sum_up01
        ←Input arbitrary number and push ‘enter’ key
```

PAD expression



In order to escape from the while statement, it is necessary for the conditional expression to be false in the iterative process.

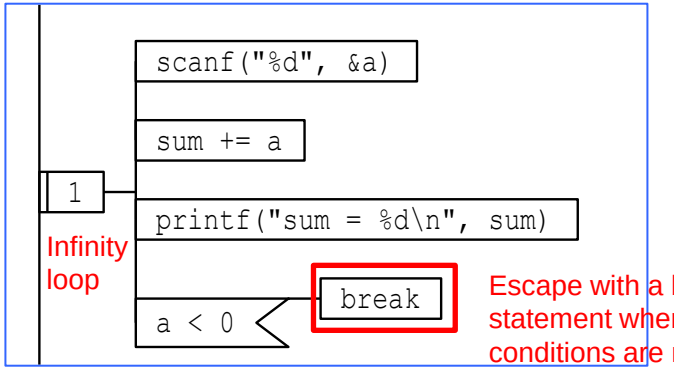
In the case of this process, the loop is exited only when sum becomes larger than 100.

Ex: Escape from while loop: `break` statement

- Sample sum_up02.c

```
/*
sum_up02: sample program of
while statement(1)
break statement
*/

#include <stdio.h>

int main()
    /**** variable declaration****/
    int a;
    int sum = 0;
    /**** processing contents****/
    while(1){ /* while statement*/

        scanf("%d",&a); /* formated input */

        sum += a;

        printf("sum = %d¥n", sum);

        if(a < 0){
            break;
        }
    }

    printf("sum is more than 100¥n");

    return 0;
}
```

```
$ gcc -Wall -o sum_up01 sum_up02.c
$ ./sum_up02
        ←Input arbitrary number
```



while (1) is often used when processing an endless loop

- Conditional expression "1" represents a true judgment
- Conditional expression "0" represents false judgment

In other words, since it is always true, it is iteratively processed forever.

To escape from an endless loop, you can use a `break` statement when certain conditions are met.

Selection process（if statement）

If statement（process when conditions are met）

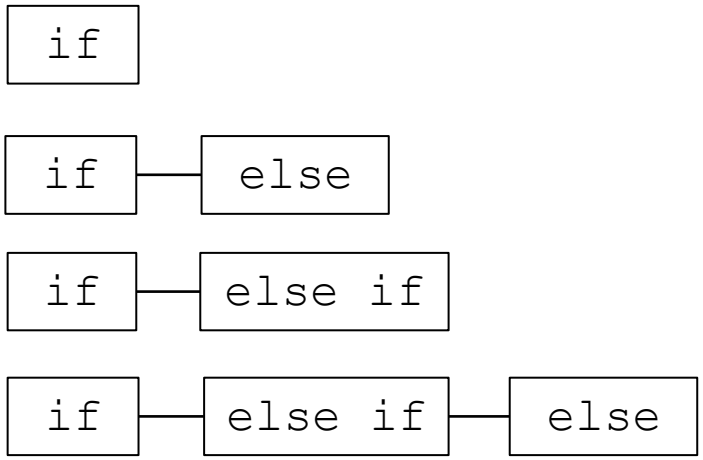
```
if （conditional expression1）{

Process-when-condition-1-is-true;

}else if（conditional expression2）{

 Process-when-condition 2-is-true;

}else if（conditional expression3）{
   •
   •
}else{

Process-for-other-condition;
}
```

If condition 1 is satisfied first, condition 2 will not be tested (only the condition that hits first is executed).

else if can be repeated any number of times

If you add else, one of the processes will always be executed.

Variations of combination

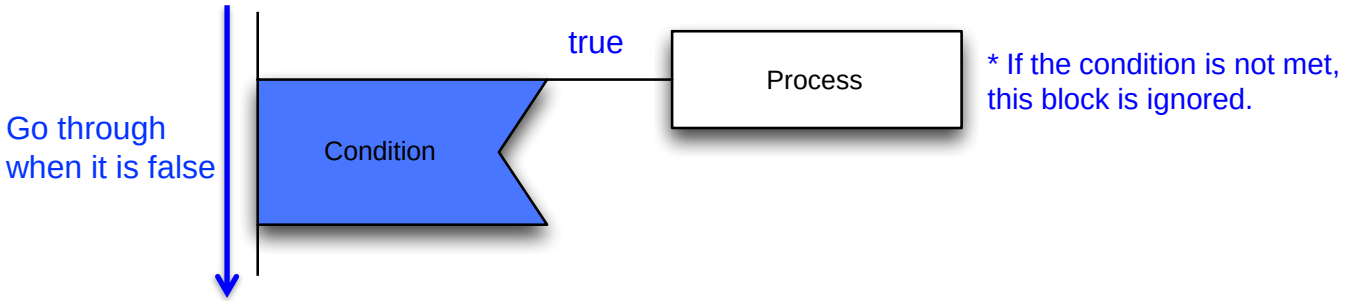


PAD expression of `if` statement

Usually the characters "true" and "false" are omitted

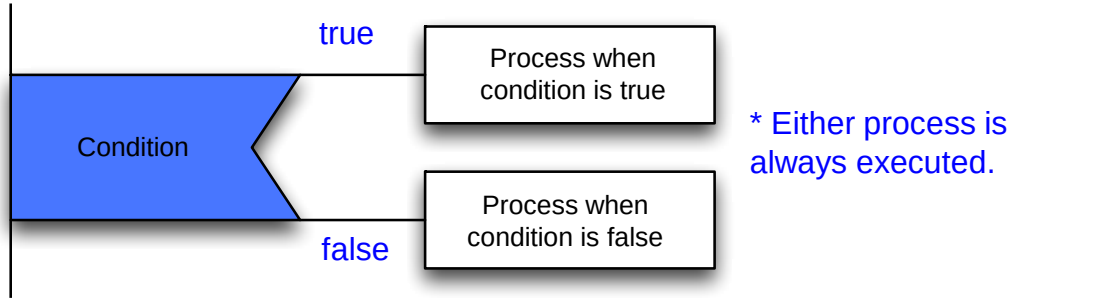
`if`

The processing content is executed only when the conditional expression is satisfied (when it is true).



`if` — `else`

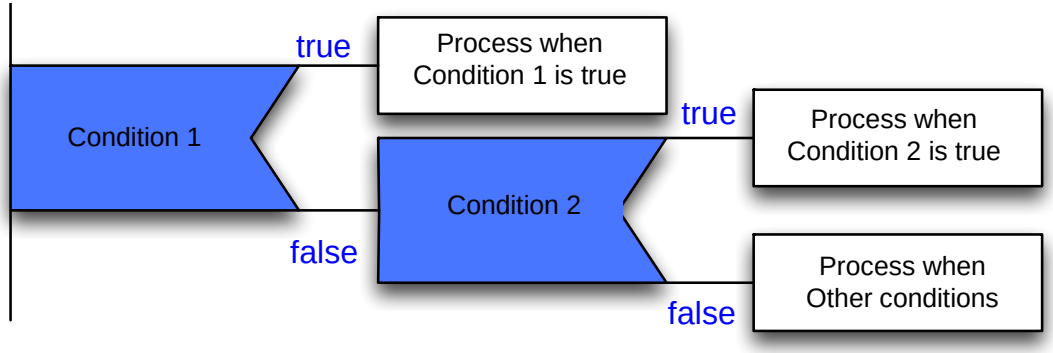
When describing both true and false processing contents for a conditional expression



`if` — `else if` — `else`

Exclusive selection from multiple conditions

(Same logic as switch statement）



（The same notation as the following `switch` statement is also possible）

Supplement: `if` – "`else if`" – `else` statement

- Exclusive selection from multiple conditions
- `if - else if -else` statements

```
if (conditional expression 1){
    Process for condition 1 is true;
}else if(conditional expression 2){
    Process for condition 2 is true;
}else{
    Process for other condition;
}
```

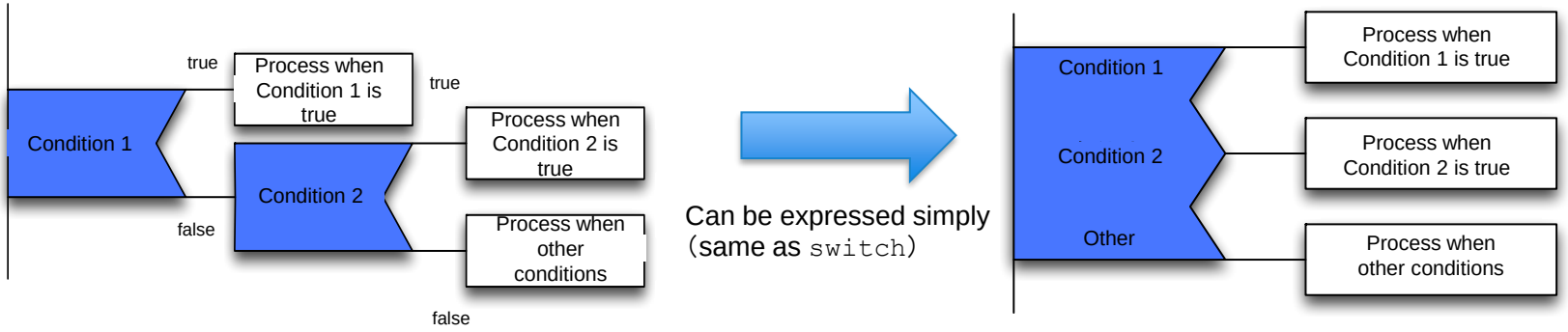
If condition 1 is met first, condition 2 will not be entered.

You can add as many `else if` statements as you like.
However, if the condition is true at the very beginning, the selection process ends at that point.

The last `else` statement is not required (If none of the conditions are met, nothing is processed)

* If there is an else at the end, one of the processes will always be executed.

PAD expression



Selection process (`switch` statement)

- Execute one out of multiple choices
  - When selecting one from the predetermined options (case) and executing it

```
switch( conditional expression ) {

    case constant 1 :
        process1 ;
        break;
    case constant 2 :
        process2 ;
        break;
    case constant 3 :
        ・
        ・
    default:
        Other conditions ;
        break;
}
```

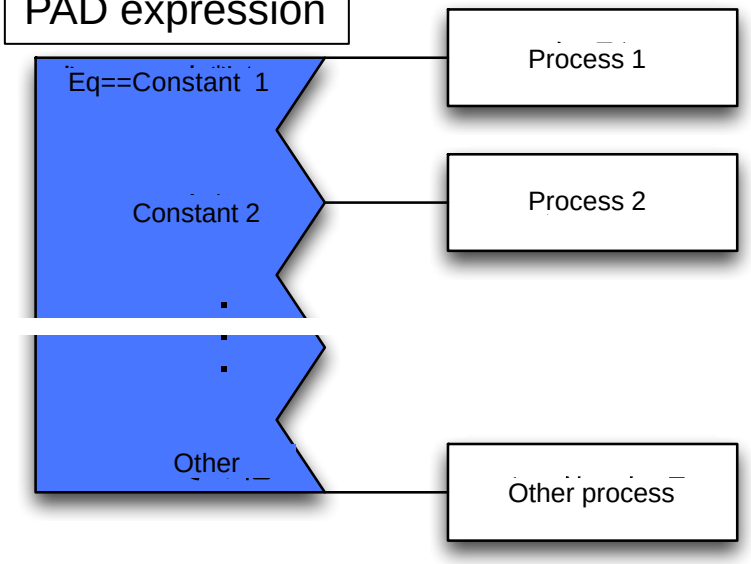
Break is required to finish processing in each case (If you don't, you will start processing the next case statement)

Can be omitted in default

Process

1. Evaluate the conditional expression and get an integer value (integer only)
2. If the integer value is the value specified in any case, it jumps to the statement following that case. If there is a break, the process ends there.
3. If not specified in any case, jump to the statement following default.
4. If default is not described, the switch statement is exited without executing anything.

PAD expression



Ex) sample of switch statement

- select_item.c

```c
#include <stdio.h>

int main(void)
    /**** variable declaration****/
    char item;

    /**** processing contents****/
    printf("Please select item [a/b/c]:");

    scanf("%c",&item);

    switch(item){ /* while statement*/
        case 'a':
            printf("a is selected¥n");
            break;
        case 'b':
            printf("b is selected¥n");
            break;
        case 'c':
            printf("c is selected¥n");
            break;
        default:
            printf("Other than a,b,c is selected¥n");
    }
    return 0;
}
```

```
$ gcc -Wall -o select_item select_item.c
$ ./select_item
```

Similarly, open select_item2.c described with the if statement with an editor and compare the structures.

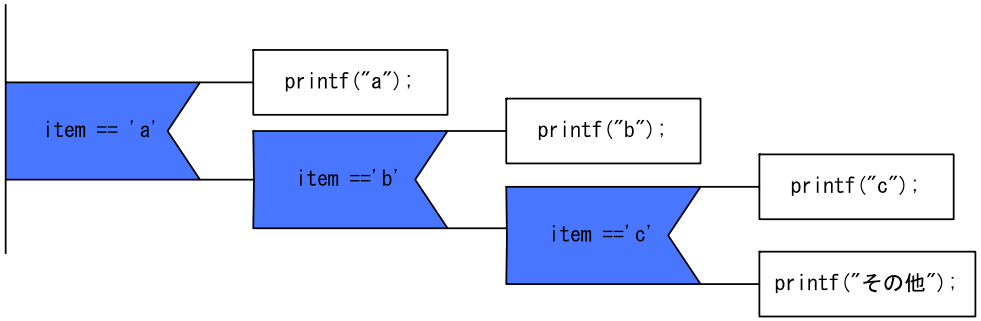
```c
    if(item == 'a'){
        printf("a is selected¥n");
    }else if(item == 'b'){
        printf("b is selected¥n");
    }else if(item == 'c'){
        printf("c is selected¥n");
    }else{
        printf("Other than a,b,c…
    }
    return 0;
}
```

Supplement: `switch` and `if` statements

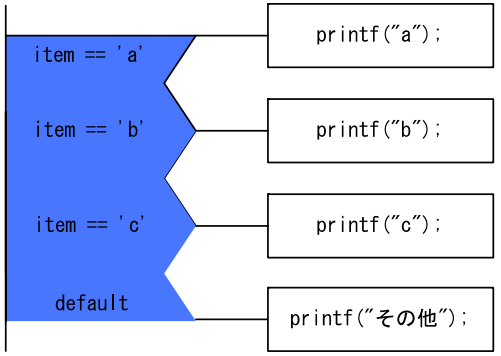
- Good! The `switch` statement has less CPU load
  - Condition judgment only needs to be done once
- Good! The `switch` statement makes it easier to understand the flow of conditions.
  - The programmer's intention becomes clear
- Bad! The `switch` statement can only handle integers
  - Characters are also a type of integer

If you want to specify a value range or condition, please use `if` statement

- Cannot handle conditional expressions such as `&&, ||`



If statement



Switch statement

Exercise 3-1: `if` statements

- abs.c:
  - Create a program that displays the absolute value of the entered real number （Tips：`if` statement）
- divisor.c
  - Create a program that determines whether B is a divisor of A for the two input integers A and B.
    - When B is a divisor of A, "B is a divisor of A." is displayed.
    - When B is not a divisor of A, "B is not a divisor of A." is displayed.

（Tips：`if – else` statement）

- rank.c
  - Create a program that distinguishes ratings of A(excellent) / B(great) / C(good) / D(bad) from the entered points and displays them. Judgment should be made as follows.
    - The score is an integer from 0 to 100
    - 0 ~ 59 → D / 60 ~ 74 → C/ 75 ~ 84 → B / 85 ~ 100 →A

（Tips：`if – if else – else` statement）

Exercise 3-2: `switch` statement

- operator.c
  - Create a calculator program that performs the four arithmetic operations (+,-, *, /) of the two input real numbers, referring to sample.c in the first lecture.
    - Input and output examples:
      - Input “12 + 3” → Display: “15.00”
      - Input “7.5 -10” → Display: “-2.50”
      - Input "2 * 5" → Display: "10.00"
      - Input “10 / 2.5” → Display: “4.00”
  - Tips
    - The operators of the four arithmetic operations are read as a char type character, and the processing is switched depending on the value in the switch statement.
    - Read 3 inputs from the console

```
scanf(”%? %? %?”, &x, &op, &y);
```

What should we choose for the format specification?

The variable op is of type char
‘+’, ‘-’, ‘*’, ‘/’ are included

Exercise 3-3: `while` statement

- even_list.c
  - Create a program that displays all positive even numbers less than or equal to the input positive integer `a`.
  - Example: Input: 13 → Output: “2 4 6 8 10 12”
  - Implement using `while` statement
- even_list_loop.c
  - Modify even_list.c to create a program that allows you to repeatedly enter positive integers from the keyboard.
  - Let the program terminate when entering a value of 0 (zero) or less.

(Note) If you cannot terminate the program, you can forcibly terminate it by pressing the ctrl key + ‘c’.

Next

- Please have break and come back to try latter half of the lecture!
- Array
  - concept
  - Subscript (index)
  - Array initialization method
  - Strings and arrays