

# **Practice of Information Processing**

(IMACU)

## **Third lecture (1st part)**

## **Previous Exercises**

---

Makoto Hirota

- Example answers of previous exercises  
(`for` statement)
- PAD expression
- Selection process (branching) format
  - `if` statement
    - `if`
    - `if else`
    - `else`
  - `switch` statement
- Iteration process (repeating) format No. 2
  - `while` statement

## ■ for statement

```
for (Initial condition ; continuation condition ; Incremental process )  
{  
    Repeating process ;  
}
```

## ■ Execution order of for statement

1. Execute the expression of the initial condition
2. Execute the processing in the for statement block
3. Perform incremental processing
4. If the continuation condition is met, return to 2.

# Previous lecture: `for` statement

4

<loop.c>

```
#include <stdio.h>

int main(){
    /*** variable declaration ***/
    int i;

    /*** processing contents***/
    for(i=0; i < 10; i++){
        printf("hello: %d\n", i);
    }

    return 0;
}
```

int i

i=0; i<10; i++    printf("hello: %d\n", i)

return 0

PAD expression

■ Exercise results

```
$ ./loop
hello: 0
hello: 1
hello: 2
hello: 3
hello: 4
hello: 5
hello: 6
hello: 7
hello: 8
hello: 9
```

# Exercise 2-4: Practice of for statement

- Modify the previous loop.c to create a program that performs the following calculation.
  - loop01.c
    - A program that calculates the sum from 1 to 100 and displays the result
  - loop02.c
    - A program that calculates the sum of numbers that are multiples of 3 from 1 to 1000 and displays the result.

# Exercise 2-4: loop01.c model answer

6

```
/*
loop01.c: sum of 1 to 100 is calculated
and the results are displayed
*/

#include <stdio.h>

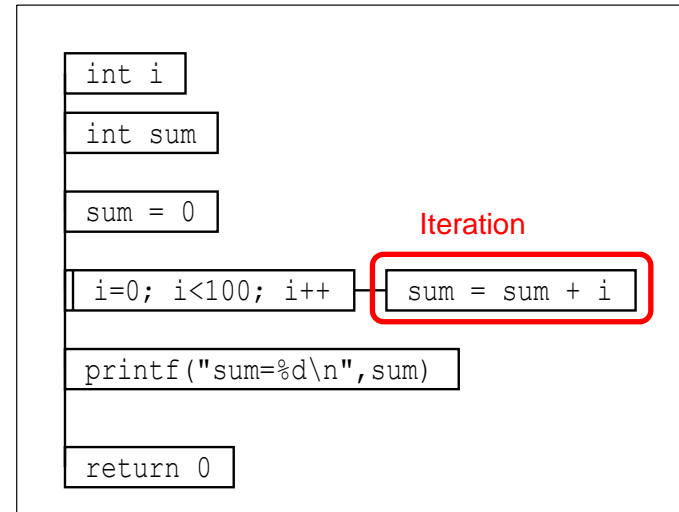
int main(){
    /*** variable declaration ***/
    int i;
    int sum;

    /*** processing contents***/
    sum = 0;

    for(i=1; i <= 100; i++){
        sum = sum+i; ← Repeating process
    }

    printf("sum=%d\n", sum);

    return 0;
}
```



PAD expression

## ❖ Memo

```
sum = sum + i;
can be written as
sum += i;
by assignment operator
```

The code is simpler because you don't need to write the variable sum twice.

# Exercise 2-4: loop01.c <Caution: Initialization>

7

```
/*
loop01.c: sum of 1 to 100 is calculated
and the results are displayed
*/

#include <stdio.h>

int main(){
    /*** variable declaration ***/
    int i;
    int sum;

    /*** processing contents***/
    sum = 0; ←variable initialization
    for(i=1;i <= 100; i++){
        sum = sum+i;
    }

    printf("sum=%d\n", sum);

    return 0;
}
```

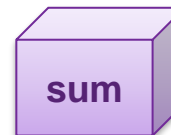
## ■ Execute results (correct)

```
$ ./loop01
sum=5050
```

## ■ Execute results (incorrect)

```
$ ./loop01
sum=37717
```

If you forget to initialize the variable, the initial value of the variable `sum` will be indefinite and you will not get the correct result.



<Review>

Declaration prepares a "box" for variable, but does not enter the contents

# Tips for not forgetting initialization (1)

8

- Get in the habit of entering initial values at the same time as variable definition

```
#include <stdio.h>

int main(){
    /*** variable declaration ***/
    int i;
    int sum = 0; ←variable initialization

    /*** processing contents***/
    for(i=1;i <= 100; i++){
        sum = sum+i;
    }

    printf("sum=%d\n", sum);

    return 0;
}
```

In C language, the initial value can be substituted when a variable is declared.

<ex>

```
int x    = 100;
float y   = 1.5;
char c    = 's';
```



# Tips for not forgetting initialization (2)

9

## ■ Add warning option `-Wall` to compiler

```
$ gcc -Wall -o loop01 loop01.c
```

You get a warning when forgetting to initialize  
(message varies depending on environment)

```
loop01-3.c:17:9: warning: variable 'sum' is uninitialized when used here
[-Wuninitialized]
    sum += i;
    ^~~~~~
loop01-3.c:12:12: note: initialize the variable 'sum' to silence this warning
    int sum;
    ^
    = 0
1 warning generated.
```

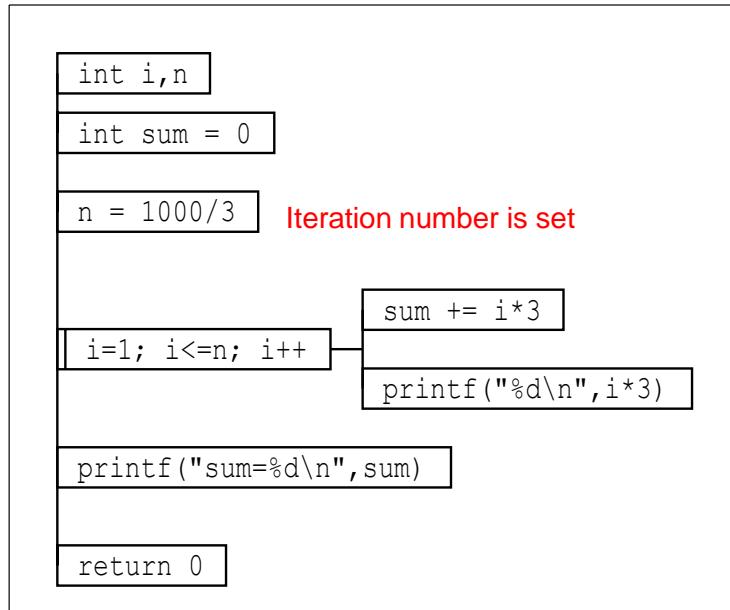
The `-Wall` option displays all gcc warnings.  
We recommend that you add this option when  
debugging your program.

However, note that it does not always warn you about absence of initialization

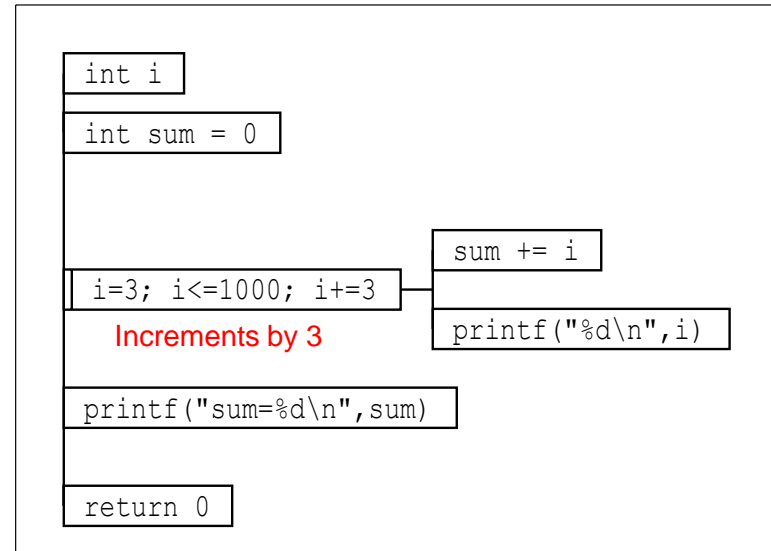
(Example) If `sum = sum + 1`, no warning may be generated.

# Exercise 2-4: PAD Expression of loop02.c

10



Example 1



Example 2

# Exercise 2-4: loop02.c model answer 1, 2

11

```
/*
sum of multiples of 3 between 1 to 1000
is calculated and the results are
displayed (ex.1 increment i with 1)
*/

#include <stdio.h>

int main(){
    /*** variable declaration ***/
    int i,n;
    int sum = 0; /* initialization */

    /*** processing contents***/
    n = 1000/3;

    for(i=1; i<=n; i++){
        sum += i*3;

        printf("%d\\n",i*3);
        /* Check adding number */
    }

    printf("sum=%d\\n", sum);

    return 0;
}
```

```
/*
sum of multiples of 3 between 1 to 1000
is calculated and the results are
displayed (ex.1 increment i with 4)
*/

#include <stdio.h>

int main(){
    /*** variable declaration ***/
    int i;
    int sum = 0; /* initialization */

    /*** processing contents***/
    for(i=1; i<=1000; i+=3){
        sum += i;

        printf("%d\\n",i);
        /* Check adding number */
    }

    printf("sum=%d\\n", sum);

    return 0;
}
```

Tip: It will be easier to debug if you use the printf function to display the progress

# Exercise 2-4: loop02.c model answer (3)

12

```
/*  
sum of multiples of 3 between 1 to 1000 is calculated and  
the results are displayed  
*/
```

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

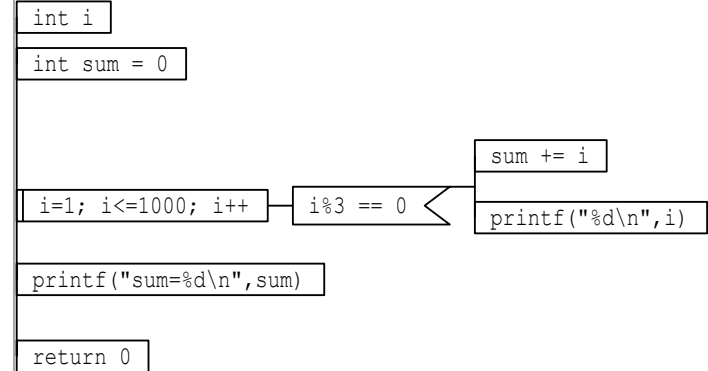
```
int main() {  
    /*** variable declaration ***/  
    int i;  
    int sum = 0;
```

```
    /*** processing contents***/  
    sum = 0;
```

```
    for(i=1; i <= 1000; i++){  
        if(i%3 == 0){/* reminder is 0*/  
            sum += i;  
            printf( "%d\\n", i); /* confirm adding number */  
        }  
    }
```

```
    printf("sum=%d\\n", sum);
```

```
    return 0;  
}
```



PAD Expression

Selection processing  
(if statement):  
If it is a multiple of 3, add it

```
$ ./loop02
```

```
sum=166833
```

# Exercise 2-5: Calculation of factorial: factorial.c

13

- Create a program that outputs the factorial ( $n!$ ) of the natural number  $n$  entered from the keyboard.

Definition of factorial

$$n! = n \times (n - 1) \times (n - 2) \times \cdots \times 3 \times 2 \times 1$$

# Hint for exercise 2-5: PAD expression of factorial.c

14

```
int num
```

```
int fact = 1
```

```
int i
```

```
printf("    Input natural number ¥n    ')
```

```
scanf("%d", &num)
```

```
i=2; i<=num; ++i    fact = i * fact
```

```
printf("%d! = %d\n", num, fact)
```

```
return 0
```

# Exercise 2-5: factorial.c model answer

15

## <1: Definition>

```
#include <stdio.h>

int main(){
    /*** variable declaration ***/
    int num; /* natural number */
    int fact = 1; /* factorial */
    int i
        Initialization

    /*** processing contents***/

    /* Input of natural number */
    printf("Input natural number =");
    scanf("%d", &num);

    /* Calculate factorial */
    for(i=2; i <= num; ++i){
        fact = i * fact;
    }

    /* Display factorial */
    printf("%d! = %d\n", num, fact);

    return 0;
}
```

(When num = 1, the contents of the for statement are not executed)

## <2>

```
#include <stdio.h>

int main(){
    /*** variable declaration ***/
    int num; /* natural number */
    int fact = 1; /* factorial */
    int i
        Initialization

    /*** processing contents***/

    /* Input of natural number */
    printf("Input natural number =");
    scanf("%d", &num);

    /* Calculate factorial */
    for(i=1; i <= num; ++i){
        fact = i * fact;
    }

    /* Display factorial */
    printf("%d! = %d\n", num, fact);

    return 0;
}
```

In the for statement,  $1! = 1 \times 1$ ,  
but it does not affect the answer.

## Exercise 2-5: Caution: Number of valid digits in the variable

16

- The output is incorrect when entering a large natural number in factorial.c ex):  $13! = 6227020800$  (correct answer)

```
$ ./factorial
Input natural number = 12
12! =479001600
$ ./factorial
Input natural number = 13
13! = 1932053504 Incorrect results
```

Scope of int type(4byte) :  $-2147483648 \sim 2147483647$  ( $\pm 2$ billion)

### ◎ How to expand the range of int type?

unsigned int type:  $0 \sim 4294967295$  (4byte, without sign)

long int type:  $-2147483648 \sim 2147483647$  (32bit OS:int type of 4byte)

64bit OS: long int type of 8byte

long long int type:  $-9223372036854775808 \sim 9223372036854775807$

```
printf("%d! = %lld¥n", num, fact);
```

long long int type

"%lld" is specified for printf of long long int type,

"%ld" is specified for printf of long int type,



## Exercise 2-6. Calculation of $\pi$ : circular.c

17

- Calculate the circular constant  $\pi$  approximately by using the following facts.

$$\arctan x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1}$$

$$\arctan 1 = \frac{\pi}{4}$$

Hints:

- $(-1)^n = 1$  when  $n$  is even.
- In the program code,  
 $1/3=0$  (integer) and  $1.0/3=0.3333333333333333$  (double).

# Exercise 2-6: circular.c model answer

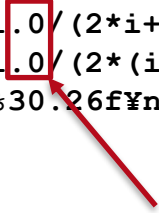
18

```
#include <stdio.h>

int main()
{
    /***** variable declaration *****/
    double ans;
    int i;

    /***** processing contents*****/
    ans=0.0;
    for (i=0;i<1e6;i=i+2){
        ans=ans+1.0/(2*i+1);
        ans=ans-1.0/(2*(i+1)+1);
        printf("%30.26f\\n",4*ans);
    }

    return 0;
}
```



This decimal is important.

3.14159165358977432447318279

This algorithm has a limitation in accuracy.  
Why?

```
float ans=1.0/3.0;
printf("%30.26f\\n",ans);
```

0.33333334326744079589843750

(7 effective digits)

```
double ans=1.0/3.0;
printf("%30.26f\\n",ans);
```

0.33333333333333331482961626

(16 effective digits)

Moreover, this round-off error accumulates due to many arithmetic calculations.

# Practice information Processing

(IMACU)

## Third lecture (2<sup>nd</sup> part)

## Selection and Iteration Process

---

Makoto Hirota

# Contents of the second part of this lecture

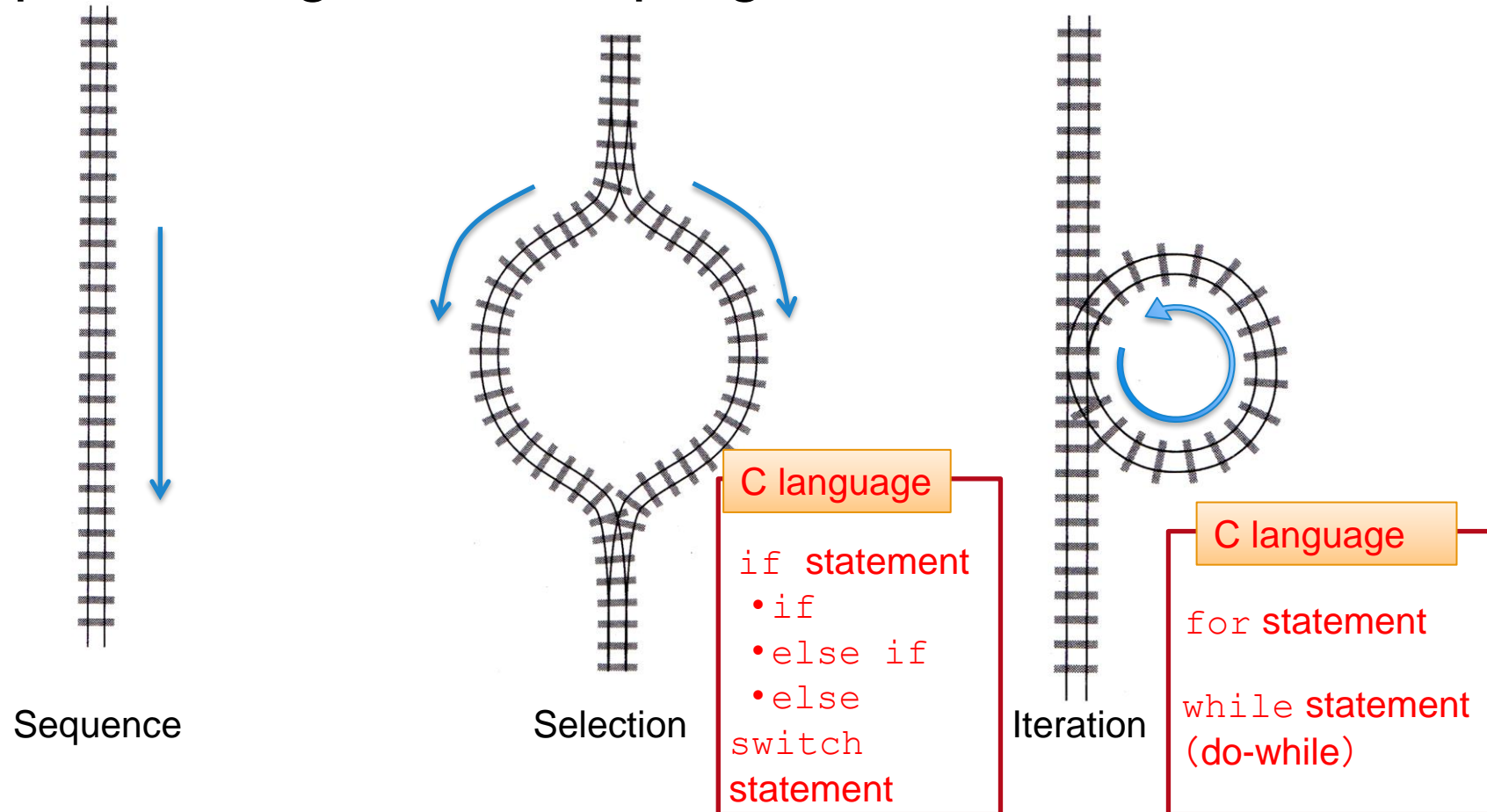
20

- Example 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

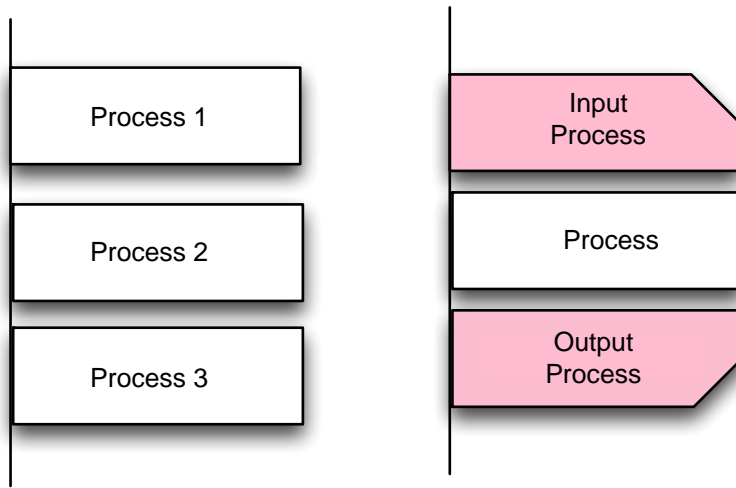
21

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



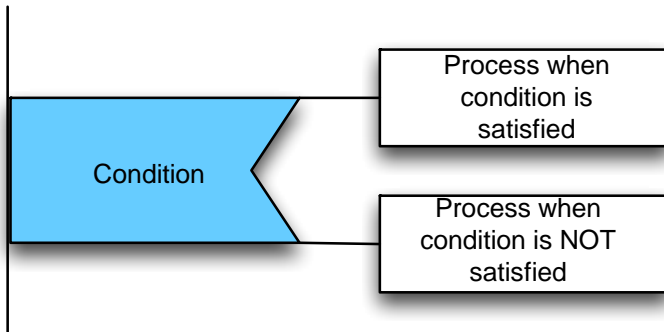
# Expression using PAD (Problem Analysis Diagram)

22

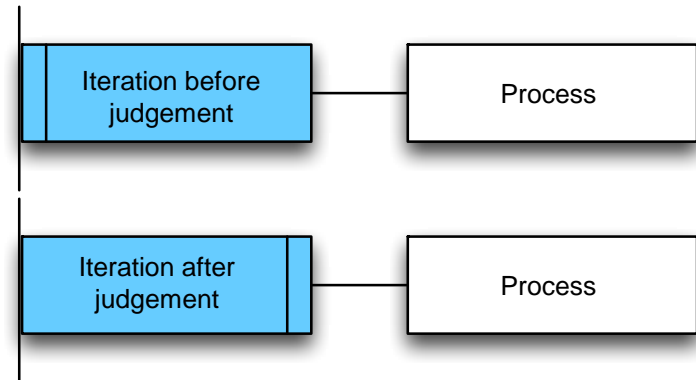


**Sequence**

Process in order of  
Top to bottom  
Left to right



**Selection**



**Iteration**

## ■ Relational operator

Operator	Meaning
$a < b$	$a < b$ (a is smaller than b)
$a \leq b$	$a \leq b$ (a is smaller than or equal to b)
$a > b$	$a > b$ (a is greater than b)
$a \geq b$	$a \geq b$ (a is greater than or equal to b)
$a == b$	$a = b$ (a is equal to b)
$a != b$	$a \neq b$ (a is not equal to b)

Operators include “=”

$\leq$   
 $\geq$   
 $!=$

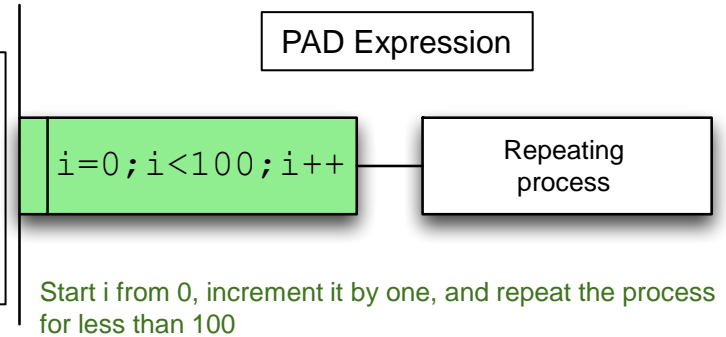
Remember  
'=' is located  
later

## ■ Logical operator (for two conditions A, B)

Operator	Meaning
$A \&\& B$	Logical product (AND): condition A and condition B are satisfied
$A    B$	Logical sum (OR): condition A or condition B is satisfied
$!A$	Negation (NOT): condition A is not met

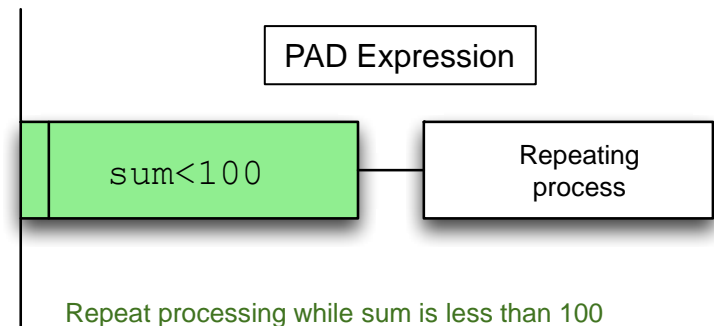
## ■ for (iteration of a certain number of times)

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



## ■ while (Repeating while the condition is satisfied) =conditional expression is true

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



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

25

## ■ Sample sum\_up01.c

```
#include <stdio.h>

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

    while(sum < 100){ /* while statement*/
        scanf("%d",&a); /* formatted input */

        sum += a;

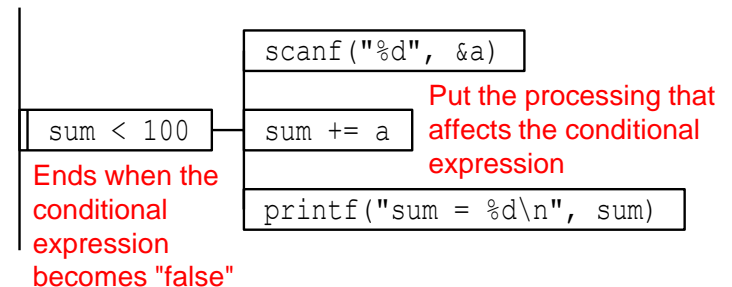
        printf("sum = %d\n", sum);
    }

    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

26

## 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); /* formatted input */

        sum += a;

        printf("sum = %d\n", sum);

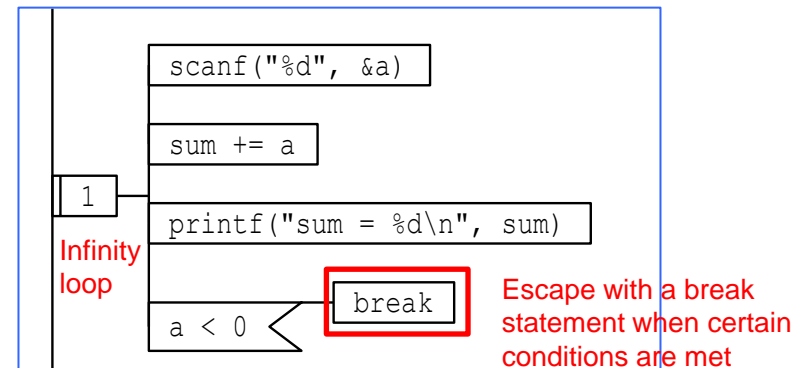
        if(a < 0){
            break;
        }

    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)

27

## ■ 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

if

if

else

if

else if

if

else if

else

# PAD expression of `if` statement

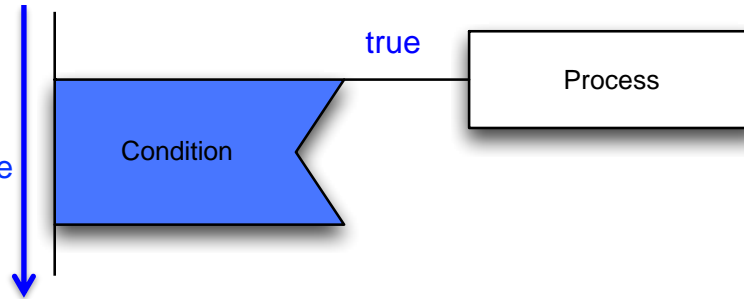
Usually the characters "true"  
and "false" are omitted

28

`if`

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

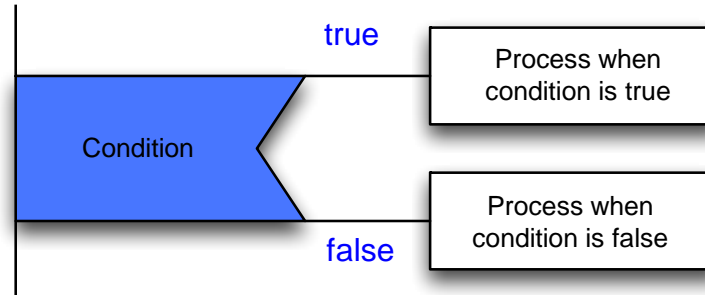
Go through  
when it is false



\* If the condition is not met, this block is ignored.

`if` — `else`

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

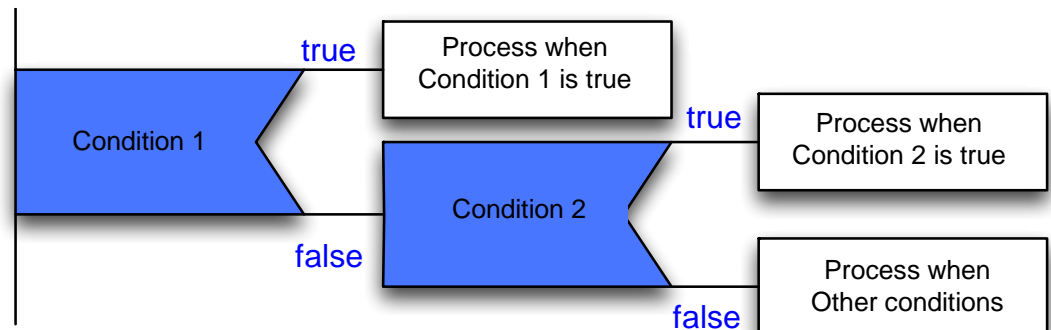


\* Either process is always executed.

`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

29

## ■ 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.

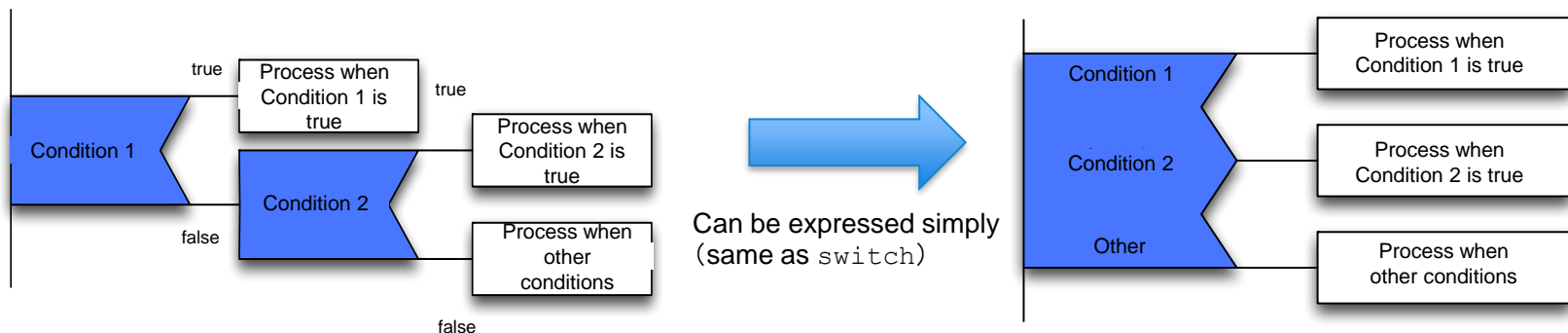
You can add as many `else if` statements as you like.

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)

30

## ■ Execute one out of multiple choices

- When selecting one from the predetermined options (case) and executing it

```
switch ( int or char variable ) {  
  
    case constant 1 :  
        process1 ;  
        break ;  
    case constant 2 :  
        process2 ;  
        break ;  
    case constant 3 :  
        .  
        .  
    default :  
        Other conditions ;  
        break ;  
}
```

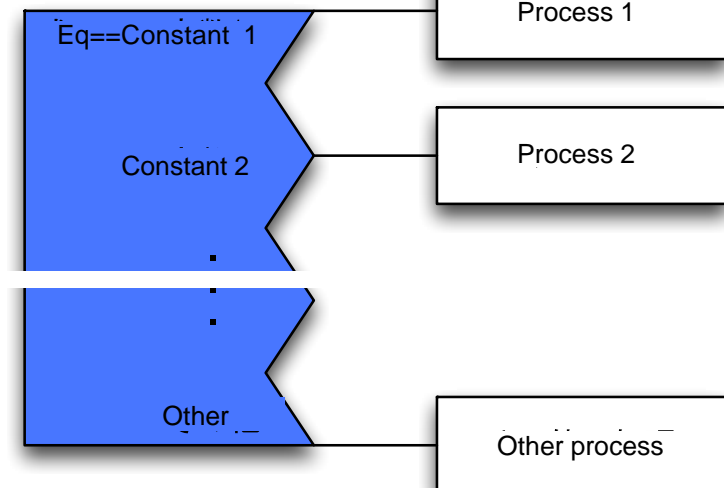
Break is required to finish processing in each case (If you don't, you will start processing the next case statement)

Can be omitted in default

## Process

1. Evaluate an integer value.
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

31

## ■ select\_item.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.

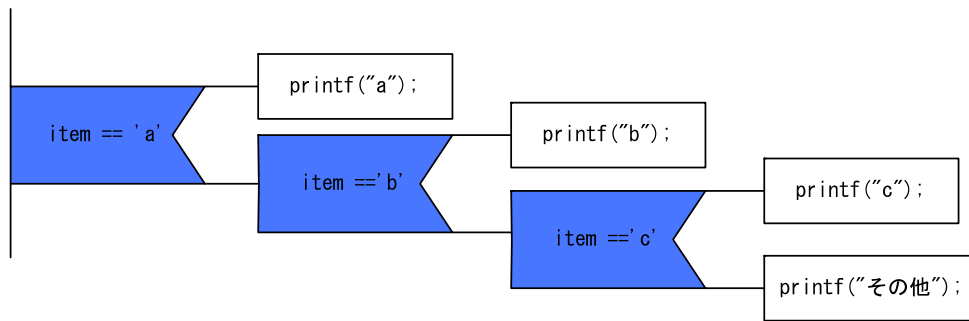
```
if(item == 'a'){
    printf("a is selected\n");
}else if(item == 'b'){
    printf("b is selected\n");
}else if(item == 'c'){
    printf("c is selected\n");
}else{
    printf("Other than a,b,c...
}
return 0;
}
```

# Supplement: `switch` and `if` statements

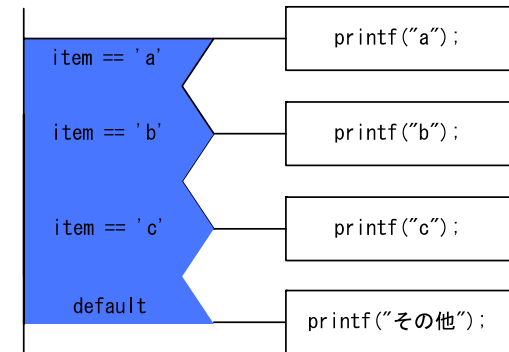
32

- **Good!** The `switch` statement has a lower CPU load
  - Condition judgment only needs to be done once.
- **Good!** The `switch` statement makes it easier to understand the conditional branching.
  - The programmer's intention becomes clear.
- **Bad!** The `switch` statement can only handle integers and characters.

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

33

## ■ `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.

## ■ `rank.c`

(Tips: `if - else` statement)

- 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
  - $0 \sim 59 \rightarrow D$  /  $60 \sim 74 \rightarrow C$  /  $75 \sim 84 \rightarrow B$  /  $85 \sim 100 \rightarrow A$

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

# Exercise 3-2: `switch` statement

34

Submission required

## ■ `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

## ■ `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’.

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

# Practice of Information Processing

(IMAC)

## Third Lecture(part 3): Array(1)

---

Makoto Hirota

## ■ Array

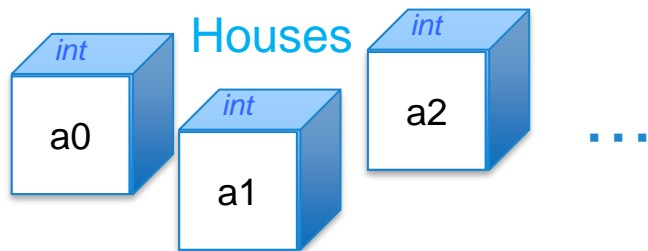
- Concept
- Index
- Initialization
- Strings and array
- Multi-dimensional array

- The way managing many variables of the same type by assigning **one name** and **consecutive numbers** (index)

Ex) When you want to prepare 100 int type variables,

## <Previous variable declaration>

```
int a0, a1, a2, ..., a99;  
a0=10;  
a1=20;
```



Schematic of single variable

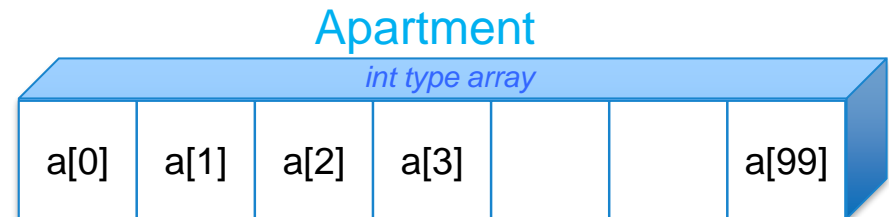
A separate data storage location (memory) is prepared for each variable



## <Variable declaration using array>

```
int a[100];  
a[0]=10;  
a[1]=20;
```

Add [] (braces) after the variable name



Schematic of array

A continuous memory space is allocated.

“Variable” has “Address” in memory space

- Specify the **number of elements** in [] when declaring an array

ex) `int a[5];` Prepare an int type array a with 5 elements

`int b[100];` Prepare an int type array b with 100 elements



Actually prepared array

`a[0], a[1], a[2], a[3], a[4];` (5 elements)

`b[0], b[1], b[2], ..., b[99];` (100 elements)

Since the subscript starts from 0, the subscript of the array with the number of elements **N** is 0 to **N-1**.

Variables `a [5]`, `b [100]` do not exist.

Be careful because it is a frequent mistake in the program!



- In the array declaration, only the memory area is reserved, and the values are indefinite.
- Like variables, each value needs to be initialized
  - If you use it without initialization, unexpected values may be used.
  - Be careful when compiling without the -Wall option
- Two ways of initialization

## (1) Assign individually

```
int a[3];  
  
a[0]=10;  
a[1]=20;  
a[2]=30;
```

for statement is very convenient when you input all zeros.

## (2) Set when declaring

```
int a[3] = {10, 20, 30};
```

Tip:

```
int a[3] = {};
```

⇒ 0 will be assigned into all variables

Note: The following assignments are not possible



```
int a[3];  
a = {10, 20, 30}; (compile error)
```

# Ex: Definition and manipulation of Array array.c<sup>42</sup>

## ■ Compile and run the sample program array.c

```
#include <stdio.h>

#define N 3 /* Macro definition */

int main(){
    /***** variable declaration****/
    int a[N];
    int i, j;

    /***** processing contents****/
    /* initialization of array */
    for (i = 0; i < N; i++){
        a[i] = 0;
    }

    /* array operation */
    a[0] = 10;
    a[1] = 20;
    a[2] = 30;

    /* array display */
    for (j = 0; j < N; j++){
        printf("a[%d] = %d\n", j, a[j]);
    }

    return 0;
}
```

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

The `for` statement is often used to manipulate arrays.

### Usage of Macro definition

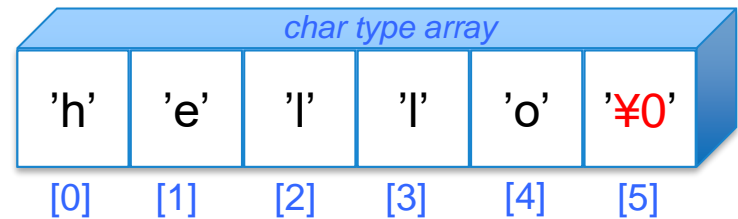
- The `#define` part at the beginning of a sentence is called a macro definition, and a specific character string (macro) in the program can be replaced with another character string.
- If you write "`#define N 3`", the preprocessor (procedure before compilation) will replace "`N`" in the program with "`3`". It is custom to write macro names in uppercase.
- The advantages of using macros are as follows.
  - You can easily change the constants related to the entire program (In this example, the number of elements in the array can be easily changed)
  - Readability is improved because programs can be written with constant names instead of numbers (In this example, using `N` instead of `3` makes it easier to understand the number of elements)

# Strings = an array of char

43

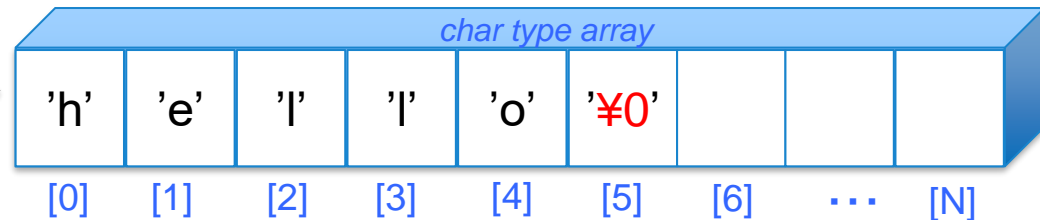
- In C language, strings are represented as char type arrays
- At the end of the string, the special character '¥0' (NUL character) indicating the end of the string is inserted.

```
char str[] = "hello";
```



\* When declaring an array, the number of elements in [] can be omitted. At this time, the memory is secured according to the number of elements given at the time of initialization.

```
char str[N] = "hello";
```



\* When a character string is assigned to a large array. (A character string can be assigned in such an assignment statement only at the time of declaration.)

In order to handle character strings with different lengths in an array with a fixed number of elements in this way, processing to detect **NUL characters** is required.

Ex) Extracts and displays character strings character by character

```
while (str[i] != '¥0') {  
    printf("%c¥n", str[i]);  
    i++;  
}
```

results



h  
e  
l  
l  
o

# Ex: Array as character strings :string.c

44

## ■ Compile and run the sample program string.c

```
#include <stdio.h>

#define LEN 100 /* Macro definition */

int main(){
    /***** variable declaration****/
    char str[LEN];
    int i;
    int n=0;

    /***** processing contents****/
    /* initialization of array */
    for (i = 0; i < LEN; i++){
        str[i] = 0;
    }

    /* input of strings */
    scanf("%s",str); ← & is not required for array

    /* display strings as array*/
    /* repeat while str[n] is not ¥0 */
    while (str[n] != '¥0'){
        printf("\str[%d] = %c¥n", n, str[n]);
        n++;
        if(n >= LEN){
            break;
        }
    }

    return 0;
}
```

```
$ gcc -Wall -o strings strings.c
$ ./strings
hello ↵ (Enter)
```

When reading a character string from the keyboard, give the array a sufficient length (LEN = 100).

Since it is not necessary to display 100 array elements, the end of the character string is determined by the '¥0'(NUL character) that is automatically inserted at the end of the input character string.

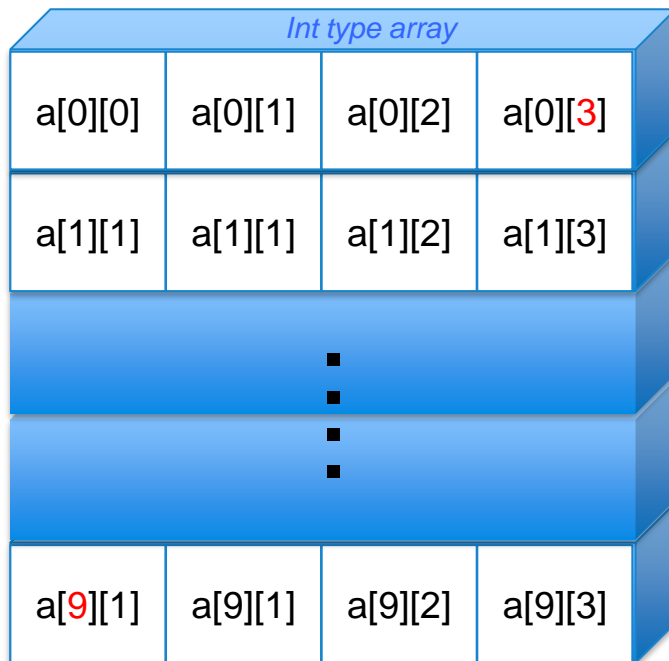
<Question>What happen when inputting “hello world”

```
$ ./strings
hello world ↵ (Enter)
```

Consider reason

Reference: Sample program string2.c uses the fgets function.

- Array with two or more subscripts
  - Effective when dealing with coordinates and spreadsheets
- A two-dimensional array
  - Example: `int a [10] [4];`



## Declaration of array

```
int array[M] [N];
```

Similar to a one-dimensional array, the subscripts of the array are the value up to M-1, N-1 for the number of elements M, N

## ■ (1) Set with declaration

```
int a[4][2] = {{1,2}, {3,4}, {5,6}, {7,8}};
```

Tip: it is sufficient to initialize with all 0s  
`int a[4][2] = {{}};` Or  
`int a[4][2] = {};`

## ■ (2) Assign individually

- A two-dimensional array is initialized with double loops (nested `for` statements).

```
int a[M][N];  
int i,j;  
for (i=0;i<M;i++){  
    for (j=0;j<N;j++){  
        a[i][j] = 0;  
    }  
}
```






Suppose M and N are defined by macros

## Exercise 3-4: Input to a one-dimensional array `array_input.c`

47

- Create a program `array_input.c` that gets 5 integers into array `a` from the keyboard, and that outputs the values of each array element and the sum..

<example>

```
$ ./array_input
10  (Enter key)
20 
30 
40 
50 
Your inputs are:
a[0]=10
a[1]=20
a[2]=30
a[3]=40
a[4]=50
Total sum is 150
```

# Exercise 3-5: Turn over a string reverse.c

48

Submission required

- Create a program reverse.c that converts the character string input from the keyboard in reverse order from the end and displays it.

<example>

```
$ ./reverse  
hello ↵ (Enter key)  
olleh
```

- If you can, make reverse.c works even when the string includes spaces. (Hint: string2.c)

Ex. “Hello World” → “dlroW olleH”



- Create a program matrix\_multiple.c that finds the product C of the 4-by-3 matrix A and the 3-by-4 matrix B.

- $C = A \times B$

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

The matrix values have already been entered in the sample source matrix\_multiple.c, so please use it.

**Tips:**

1. Be aware of which is the row and which is the column when coding  
`int A[4][3]`  
row column
2. Matrix subscripts start at 0

<Result>

```
$ ./matrix_multiple
C[0][0] = 14
C[0][1] = 20
C[0][2] = 26
C[0][3] = 32
C[1][0] = 35
C[1][1] = 50
C[1][2] = 65
C[1][3] = 80
C[2][0] = 14
C[2][1] = 20
C[2][2] = 26
C[2][3] = 32
C[3][0] = 35
C[3][1] = 50
C[3][2] = 65
C[3][3] = 80
```

```
$ ./matrix_multiple
C =
| 14  20  26  32|
| 35  50  65  80|
| 14  20  26  32|
| 35  50  65  80|
```

- You can easily save the output of the program to **a text file** by using the method "Redirect (>)" on the console terminal.
  - The results of printf etc. are displayed in the standard output. By default, the console terminal is the standard output destination.
  - Redirection is a method of changing the standard output destination of a program to a file.

```
$ ./matrix_multiple > result.txt  
$ cat result.txt ← cat is command to output the contents
```

The output contents of the program are saved in a file named "result.txt".

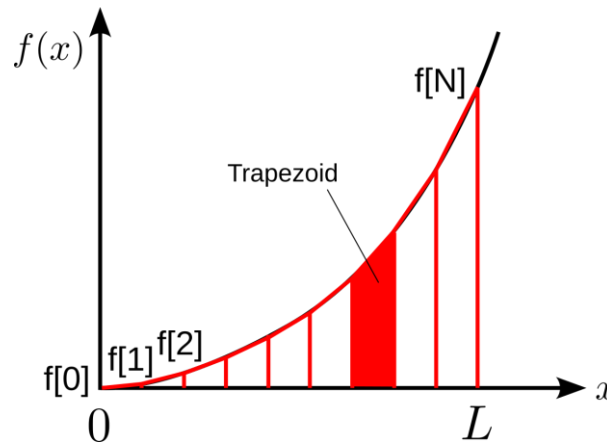
# Exercise 3-7: Numerical integration trapez.c

51

Submission required

## ■ Create a program trapez.c in which

1. The values of a function  $f(x) = x^2$  at  $x = iL/N$  ( $i = 0, 1, 2, \dots, N$ ) are stored in an array  $f[0], f[1], f[2], \dots, f[N]$ .  
(Choose a real number  $L > 0$  and an integer  $N > 0$  freely)
2. The integral  $\int_0^L f(x)dx$  is calculated approximately by using the trapezoidal rule.



## ■ Check the error of the result in comparison with the analytical answer.

- Review of Previous Lectures
- Selection and iterative processes
  - Selection process: if, switch statements
  - Iterative process: for, while statements
- Array
  - Concept
  - Index
  - Initialization of array
  - Strings and array
  - Multidimensional array

- Example answer of exercise
  - Exercise of array
  - Midterm report
- 
- See you next week!