

# Introduction to Algorithms and Data Structures

## Lesson 1: Foundation of Algorithms (1) Basic Models

Professor Ryuhei Uehara,  
School of Information Science, JAIST, Japan.

[uehara@jaist.ac.jp](mailto:uehara@jaist.ac.jp)

<http://www.jaist.ac.jp/~uehara>

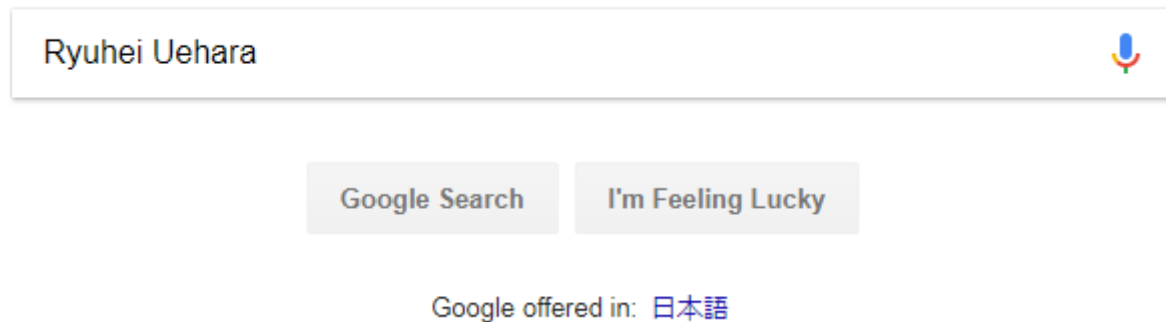
# Summary

## Introduction to Algorithms and Data Structures

- By Professor Ryuhei Uehara, JAIST
- Goal: Understanding of *value* of **Algorithms**
  - An **algorithm** is a **way/method for solving a problem**.
  - A **data structure** is a **way/method for storing data** in a computer.
  - In general, for a problem, there are many combinations of algorithms and data structures. We need to evaluate them according to their efficiency, and choose the best one.
  - However, the important point is that to master the way of thinking of algorithm design.
  - In this short course, we learn several basic and representative problems and algorithms for them. We analyze their correctness and efficiency.

# Some information and materials

- <http://www.jaist.ac.jp/~uehara/course/2018/myanmar/>
  - Please check it at least once 😊
  - You can find the page by google;



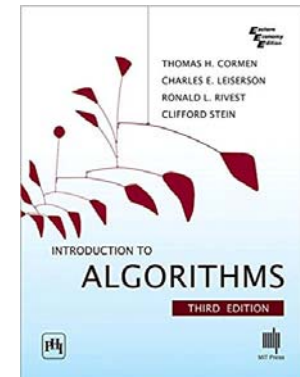
→ [Courses](#) →

Short Lectures in University of Information Technology, Yangon, Myanmar.

From January 22 (Monday) to January 26 (Friday) 2018: [Introduction to Algorithms and Data Structures.](#)

# References

- Textbook
  - “Introduction to Algorithms, 3<sup>rd</sup> ed.”  
Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, 2010, MIT Press.
  - “First Course in Algorithms through Puzzles (tentative),” Ryuhei Uehara, in preparation, a publisher in US.



# Requirements

- No special knowledge is required, but...
  - It is better to have some experience of programming
    - ...in any programming language
      - C, C++, Java, C#, Ruby, Python, Scheme, Haskell, ...
  - **Algorithm** itself is independent from any programming language.
    - I will use so-called “pseudo-code” to describe high-level idea of an algorithm.

# What algorithm is...

An abstract description of method for solving a problem using a computer.

- What “solving a problem” means;
  - We can obtain a correct answer for any input
  - It can be obtained with reasonable costs;
    - Computation is done in a polynomial time of the length of an input
    - in a polynomial space (=memory) of the length of an input
- A problem is “unsolvable” if
  - it takes so long time for some inputs,
  - it takes so much memories for some inputs, or
  - (we cannot make any program for the problem)

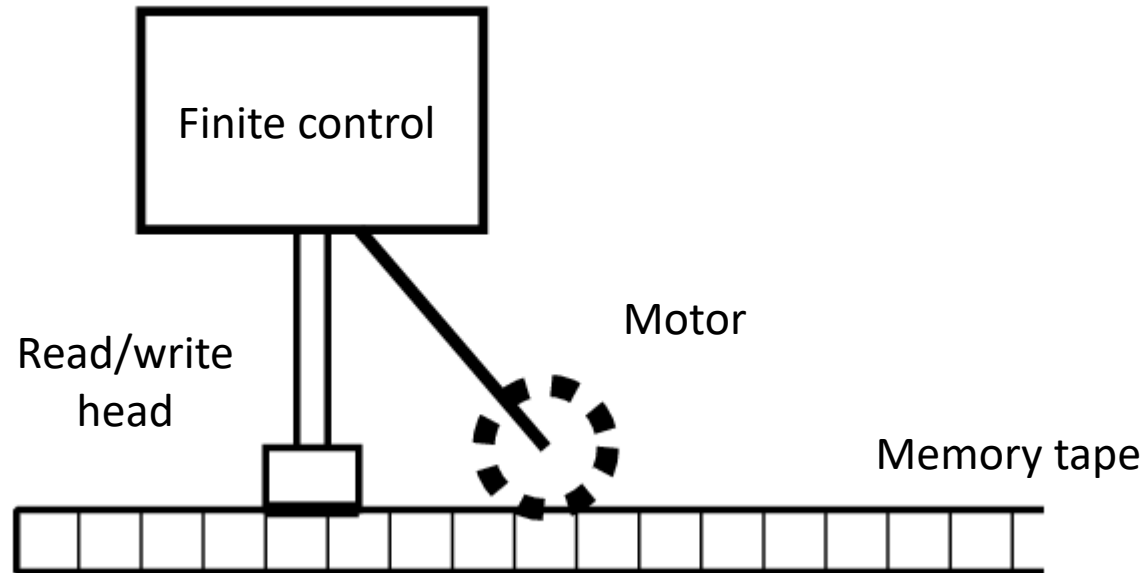
# Model of Computing

How does “computer” work?

What is a “computation”?

- Description/efficiency of an algorithm are different depending on a model of computation.
  - What “basic operations” are?
  - What kind of data in memory?
    - Natural numbers, real numbers (with infinite accuracy?), images, music data...?
- There are some standard models of computing
  - **Turing machine**: The mathematical model by Alan Turing. Base of all arguments of computation.
  - RAM model: a standard model when we consider algorithms.

# Turing machine model

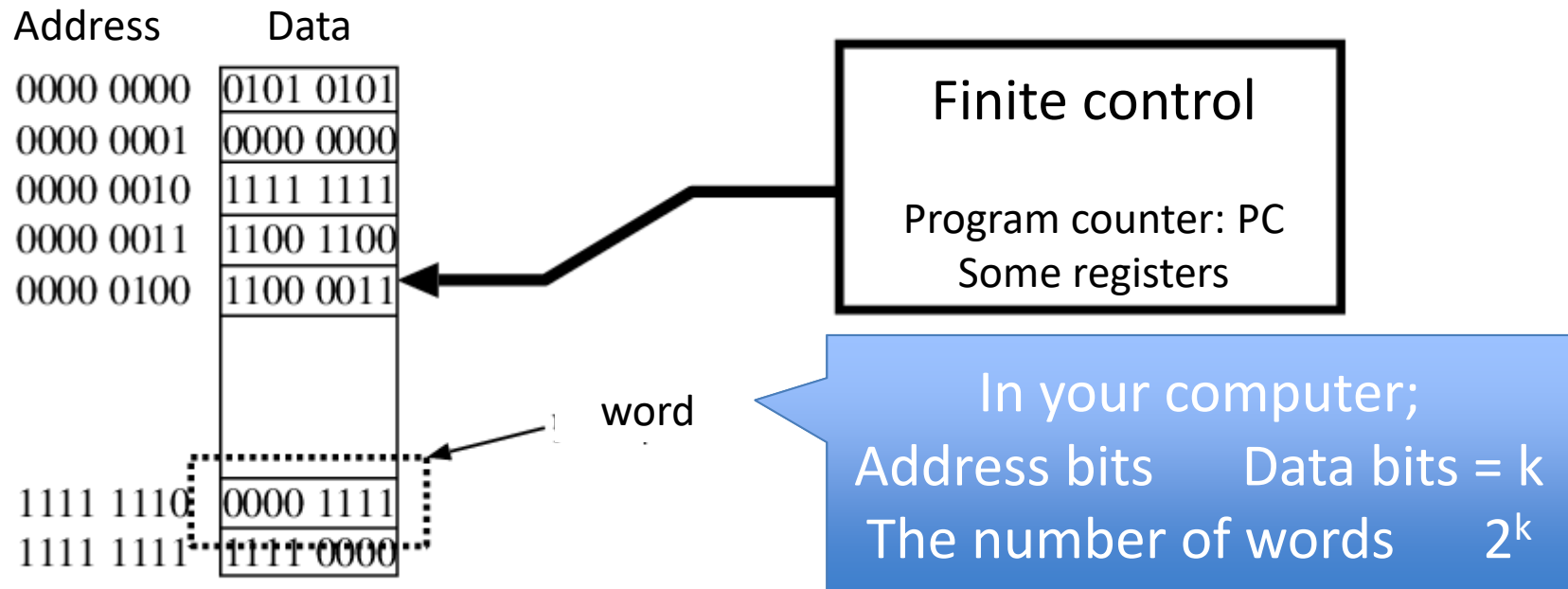


- Quite simple mathematical/theoretical model.
- Turing prove that a Turing machine is “universal”, which means that every computable function can be computed by a Turing machine.
- Turing machine is toooooo simple to do programming in a real world
  - Few basic operations like  $+$ ,  $-$ ,  $*$ ,  $/$ , and so on...
  - It is not good for discussion of “algorithms”



# RAM Model

## (Random Access Memory)



- It consists Memory and CPU (Central Processing Unit)
  - We do not mind Input/Output
- It is essentially the same as your computer
- CPU can access any address randomly (not sequentially) in a unit cycle
- Programming language C is a system that show you this structure implicitly (like arrays and pointers)

# Programming Language

- Compiler translates any “readable” program (for human) to an executable file in machine language (for the CPU)
- E.g. Programming language C; It is okay if you know...
  1. variable
  2. array
  3. pointer
  4. control statement (e.g., if, while)
  5. recursive call

# Basic of C: Hello World

- Display “Hello World” on screen

```
#include <stdio.h> /* for printf*/  
  
main(){  
    printf(“Hello World”);  
}
```

statement

Semi-colon at  
the end of a  
statement

“Algorithms” do not depend on programming language,  
but we need some agreement in this class.

# Basic of C: Mathematics

- Mathematical operations: +, -, \*, /

Equation	meaning
3+4	Add 3 and 4
3-1	Subtract 1 from 3
3*3	Multiply 3 and 3
4/2	Divide 4 by 2

- We do not mind if they are integers (`int`, etc.) or real numbers (`float`, `double`, etc.) in this class

## Note: For C beginner

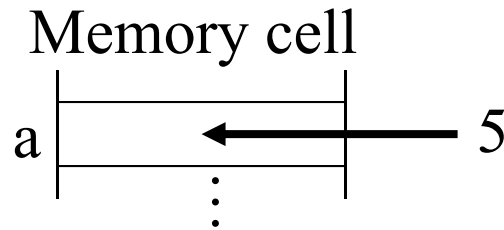
- integer/integer is an integer
  - Ex:  $1/3$  is  $0$ , and  $1.0/3$  is  $0.3333\dots$
- You can use  $()$  for control of the order of operations
  - You cannot use  $\{\}$  and  $[]$  in mathematical formula
  - Ex:  $\{(3+4)*3+4\}*6$  is not valid. You have to write  $((3+4)*3+4)*6$
- No power operations (you have some library of functions to compute it)

# Basic of C: Variable

- Variable: It is a memory cell, that indicates the “place” to memory a result of computation
- Rules for naming
  - Start with alphabet (UPPER, lower letters, and \_)
  - From the second letter, you can use alphabets and numbers
    - Not any other
  - Upper and lower letters are different
    - FF, ff, fF, and Ff are all different names
  - Not reserved words in C (e.g., main, include, return)
  - Good: x, orz, T\_T, IE9, projectX, ff4, y2k, JAIST
  - Bad: 7th, uehara@jaist, ac.jp, tel#

# Basic of C: Assignment statement

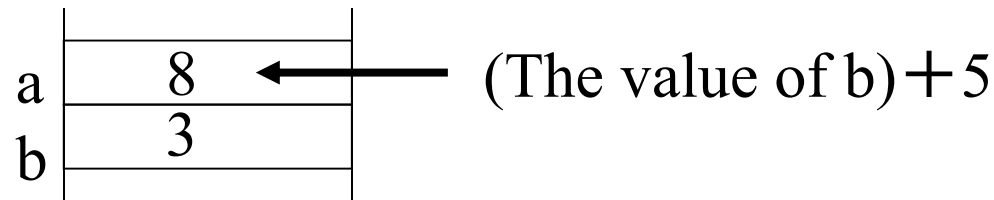
- $a=5$



“=” is not “equal” in the sense of mathematics

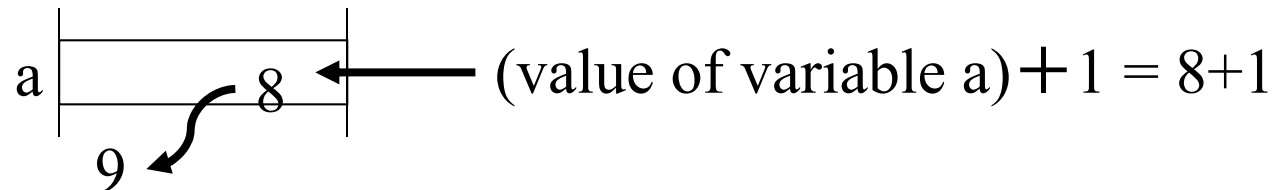
- Store the value 5 to the place named by a in memory

- $a=b+5$



- Store value of “value stored at the place named by b (or value of the variable b) plus 5” to the place named by a

- $a=a+1$



- Store value of “the value of variable a plus 1” to the place named by a

# Basic of C: declaration of variable

- You have to declare variables beforehand (in C language)

Good

```
main(){  
    int a,b;  
    a = 5; b = 3;  
    printf("a+b=%d",a+b);  
}
```

Variables a and b in integer

Bad

```
main(){  
    a = 5;  
    printf("%d",a);  
}
```

It is not good!

Note: Recent language (like python) does not require to declare beforehand.



# Basic of C: Mathematical functions

	function	Math. symbol	type	Parameter type
Square root	<b>sqrt(x)</b>	$\sqrt{x}$	<b>double</b>	<b>double</b>
Power	<b>pow(x, y)</b>	$x^y$	<b>double</b>	<b>double</b>
Logarithm	<b>log(x)</b>	$\log_e x$	<b>double</b>	<b>double</b>
Logarithm	<b>log10(x)</b>	$\log_{10} x$	<b>double</b>	<b>double</b>
Exponential	<b>exp(x)</b>	$e^x$	<b>double</b>	<b>double</b>

- Source code: include the following header file  
`#include <math.h>`
- Compile: Option `-lm` is required
  - `gcc main.c -lm`

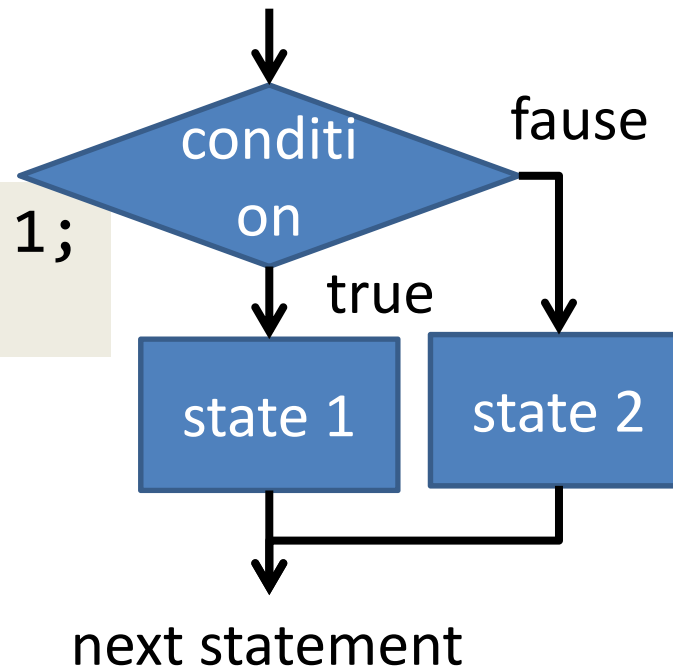
# Basic of C: Control statements

## if statement – conditional branch

- Grammar

```
if (condition) state 1;  
else state 2;
```

If condition is true, perform statement 1, and perform statement 2 if it is false



– Ex: Output EVEN if n is even, and ODD if it is odd.

```
if(n%2==0) printf("EVEN");  
else printf("ODD");
```

# Basic of C: Representations of conditions (1/2)

symbol	meaning	example	meaning of example
==	equal	<code>n == 2</code>	n is equal to 2
!=	not equal	<code>n != 0</code>	n is not equal to 0
>	greater than	<code>n &gt; 3</code>	n is greater than 3
>=	g.t. or equal	<code>n &gt;= 3</code>	n is g.t. or equal to 3
<	less than	<code>n &lt; 0.01</code>	n is less than 0.01
<=	l.t. or equal	<code>n &lt;= 0.01</code>	n is l.t. or equal to 0.01
&&	and	<code>0 &lt; n &amp;&amp; n &lt;= 10</code>	n is greater than 0 and less than or equal to 10
	or	<code>n &lt; 0    0 &lt; n</code>	n is less than 0 or greater than 0
!	not	<code>!(n &lt; 0.01)</code>	n is not less than 0.01

# Basic of C: Representations of conditions (2/2)

- You cannot compare 3 or more items
  - $0 < x < 5$        $\rightarrow$   $0 < x \ \&\& \ x < 5$
  - $a == b == c$      $\rightarrow$   $a == b \ \&\& \ b == c$
- Example: Check of the leap year
  - Dividable by 400, or
  - Not dividable by 100 but dividable by 4

```
year%400==0 || (year%100!=0 && year%4==0)
```

# Basic of C: Control statements

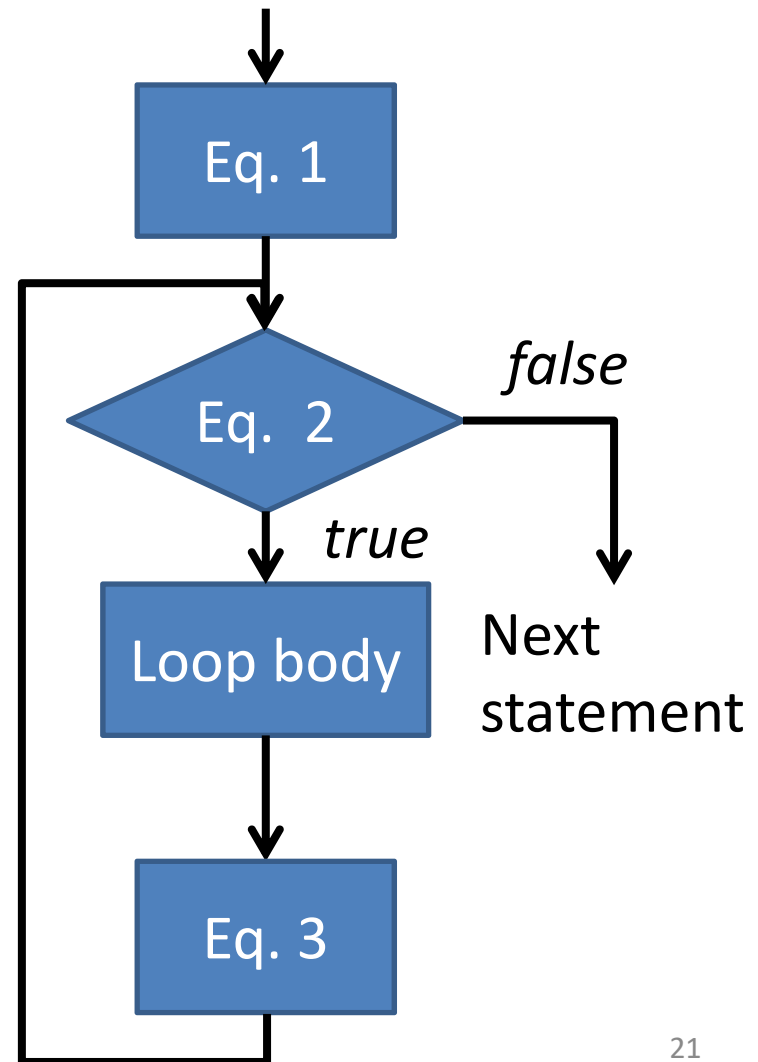
## for loop – repeating (1/4)

- Grammar

```
for(eq.1;eq.2;eq.3){  
  loop body  
}
```

- It runs as follows:

- A) Execute eq. 1
- B) If eq.2 is *true*, step C, and step D if *false*
- C) Perform loop body and eq. 3, jump to B
- D) Go to next statement



# Basic of C: Control statements

## for loop – repeating (2/4)

Example: Output the sum  $\sum_{i=1}^n i$  between 1 to n

```
int i,n,sum;
n=/*initialized somehow*/;
sum=0;
for(i=1;i<=n;i=i+1){
    sum=sum+i;
}
printf("1+...+%d=%d",n,sum);
```

# Basic of C: Control statements

## for loop – repeating (3/4)

Example: Output the sum  $\sum_{i=1}^n i^2$  between 1 to n

```
int i,n,sum;
n=/*initialized somehow*/;
sum=0;
for(i=1;i<=n;i=i+1){
    sum=sum+i*i;
}
```

# Basic of C: Control statements for loop – repeating (4/4)

- Ex: Compute  $\sum_{i=1}^n (2i - 1)^2$

```
int i,n,sum;
n=/*initialized somehow*/;
sum=0;
for(i=1;i<=2n-1;i=i+2){
    sum=sum+i*i;
}
```

i indicates 2j-1

- Why is this correct?

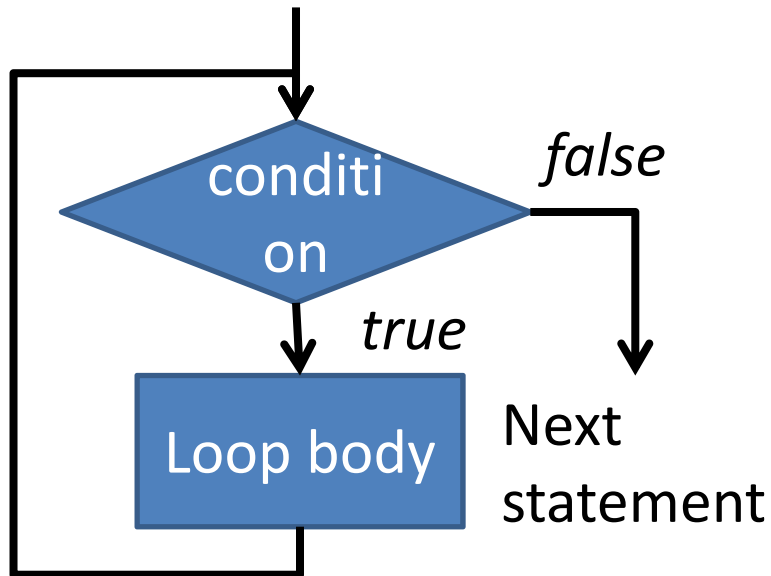
– Because;  $\sum_{i=1}^n (2i - 1)^2 = 1^2 + 3^2 + \dots + (2n - 1)^2$



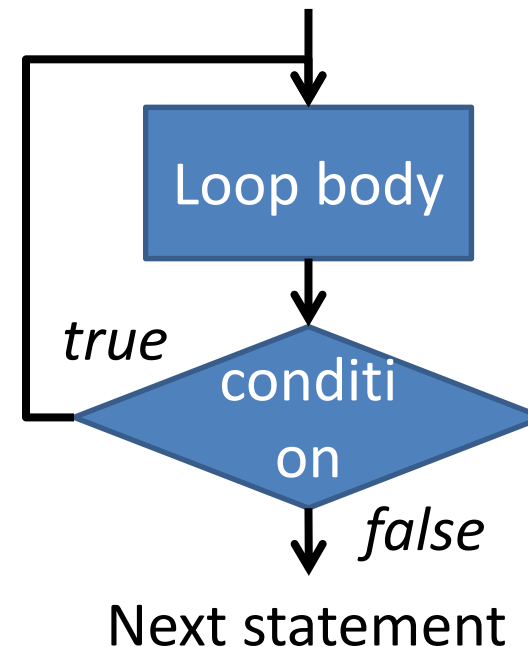
# Basic of C: Control statements while loop & do-while loop (1/2)

- Grammar

```
while(condition){  
    loop body  
}
```



```
do{  
    loop body  
}while(condition)
```



# Basic of C: Control statements

## while loop & do-while loop (2/2)

Ex: Compute GCD(a,b) of two integers a and b

```
int a,b,r;
a=/*some value*/;
b=/*some value*/;
do{
    r = a % b;
    a = b; b = r;
}while(r!=0);
printf("G.C.D.=%d",a);
```

Ex: a=1848, b=630

a	b	r=a%b
1848	630	588
630	588	42
588	42	0
42	0	0

This method (algorithm) is known as  
“Euclidean mutual division method”

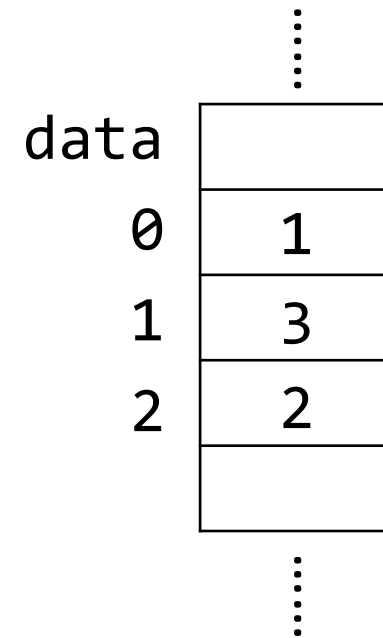
# Basic of C: Array (1/2)

- What is array?

Data structure that aligns many data in the same type (int, float, etc.) sequential in memory

- Ex: `int data[3]`
  - 3 consecutive memory cells are kept as name “data”, in which each cell stores an integer.

```
int data[3];  
data[0]=1;  
data[2]=2;  
data[1]=3;
```



# Basic of C: Array (2/2)

## Get the maximum

- Ex: compute the maximum value in integer data[100]

```
int data[100];
int i,max;
/*data is initialized somehow*/
max=0;
for(i=0;i<100;i=i+1){
    if(max<data[i]) max=data[i];
}
printf("maximum data = %d",max);
```



Wrong!

Q: Is this program correct?

# Basic of C: Array (2/2)

## Get the maximum

- Ex: compute the maximum value in integer data[100]

```
int data[100];
int i,max;
/*data is initialized somehow*/
max=0;
for(i=0;i<100;i=i+1){
    if(max<data[i]) max=data[i];
}
printf("maximum data = %d",max);
```

Wrong!

When all data is negative, it outputs 0 as the maximum!

Q: Is this program correct?

# Basic of C: Array (2/2)

## Get the maximum

- Ex: compute the maximum value in integer data[100] – make it correct

```
int data[100];
int i,max;
/*data is initialized somehow*/
max=data[0];
for(i=1;i<100;i=i+1){
    if(max<data[i]) max=data[i];
}
printf("maximum data = %d",max);
```

The value of max is always in data

# 10 minutes exercise

- I'll collect them
  - I'll just check if you understand the topic.
- 5 minutes exercise
  - What does this function  $S(x,y)$  do?

```
S(int x, y) {  
    x=x+y;  
    y=x-y;  
    x=x-y;  
}
```

- Write your student ID, name, and answer (and any comment is welcome 😊)