# I111E Algorithms & Data Structures
# 1. Basic Programming

School of Information Science

Ryuhei Uehara & Giovanni Viglietta

uehara@jaist.ac.jp & johnny@jaist.ac.jp

2019-10-16

All materials are available at

http://www.jaist.ac.jp/~uehara/couse/2019/i111e

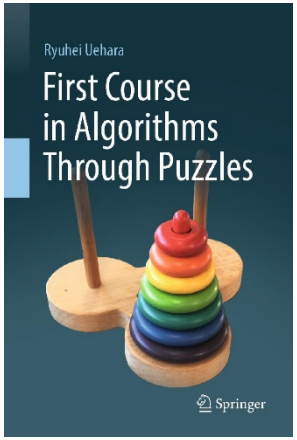
# Summary

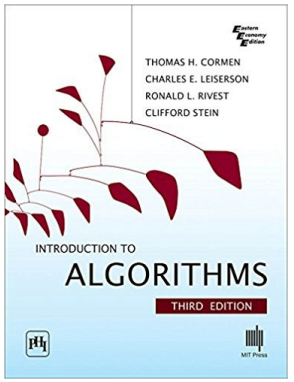
- I111 Algorithms and Data Structures
- Lecturers: Ryuhei Uehara & Giovanni Viglietta
- Goal: To understand the meaning and importance of algorithms.

A formal procedure for solving a problem is called an algorithm and a way of storing data in a computer is called a data structure. There may be a number of combinations of algorithms and data structures for a problem, in general. It is important to evaluate them by computation time and space requirement to choose the best combination. It is not sufficient to understand conventional algorithms, but it is more meaningful to master how to design algorithms. In this lecture, a general but basic scheme for algorithm design through validation of the correctness of algorithms and investigation of improvement of algorithm efficiency is explained.

# References

- Textbooks
  - "Theory of Algorithms," Tetsuo Asano, Koichi Wada, Toshimitsu Masuzawa, 2003, Ohm Publishing Co. (in Japanese)
  - "First Course in Algorithms through Puzzles," Ryuhei Uehara, 2019, Springer.
  - "Introduction to Algorithms, 3rd ed." Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, 2010, MIT Press.




We do not necessarily follow the textbooks,,,

# Evaluations

- Viewpoint of evaluation :
  - Comprehension of theory and implementation of algorithms and data structures.
- Evaluation method :
  - Reports (40pts) and examination (60pts)
    - I’m now planning 2 reports with final examination.

# Schedule of Lectures

- Examination: 12/4?

- Wednesdays (10:50-12:30)
  - 10/16, 10/23, 10/30, 11/06, 11/13, 11/20, 11/27
- Mondays (09:00-10:40)
  - 10/21, 10/28, 11/11, 11/18, 11/25, 11/28, 12/02

- Tutorial Hours (13:30-15:10) on Mondays
  - Basically, you can ask us at our office (I67).
  - Sometimes, we will give supplemental lectures, e.g., answers and comments on reports, etc.

# Requirements

- Lectures are given in English
- You can ask/answer in English or Japanese

- Note that “algorithm” and “programming” are different. “programming” is implementation of algorithm.

- We do not assume any specific language, but we use C as an example.

- You can use any programming language such as c, C++, Java, Delphi,,,, perl, ruby, python, basic… in your reports.

# What's an algorithm?

Algorithm = Description of a method of solving a problem using computer

- What's a good algorithm?
  - It outputs a correct answer for any input
  - It outputs an answer in reasonable cost
    - polynomial time of input length
    - polynomial space of input length
- What's a bad algorithm?
  - It takes a loooong time for some input
  - It uses a huuuge memory for some input
  - (There exists unsolvable problems by any program)

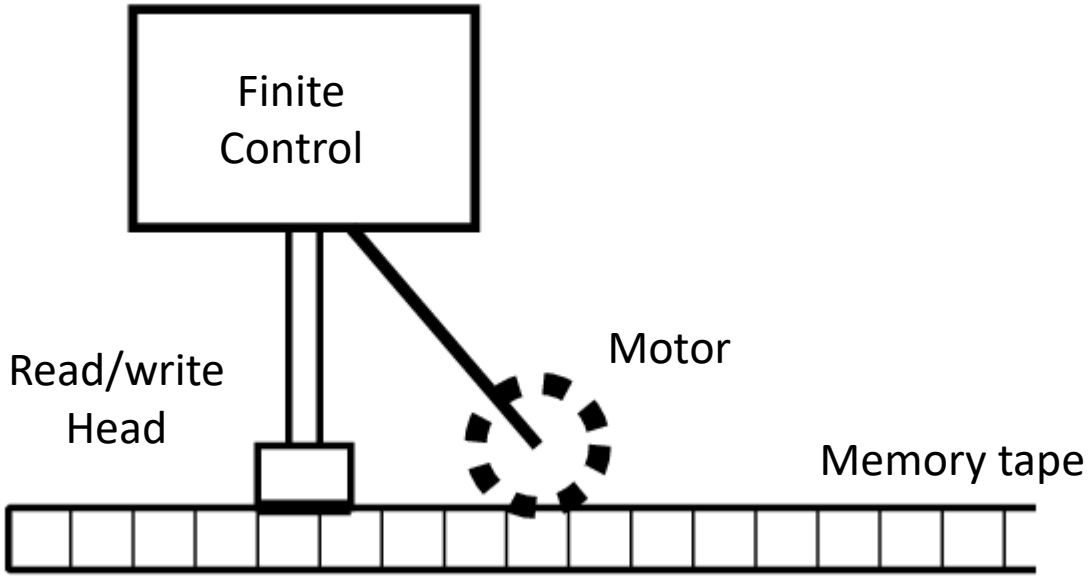
# Models of “computation”

How can we evaluate time and space?

→ First of all, how do computers work?

- Efficiency of algorithms may change according to computation model
  - What are “basic operations”?
  - What kind of data can it store?
    - Natural numbers, real numbers (?), images, musics…?
- There are some standard models
  - Turing machine: That Turing innovated. Base of all computation models.
  - RAM model: Standard model for algorithm theory.
  - We may use models based on GPU and/or quantum computation in near future…

# Turing Machine Model

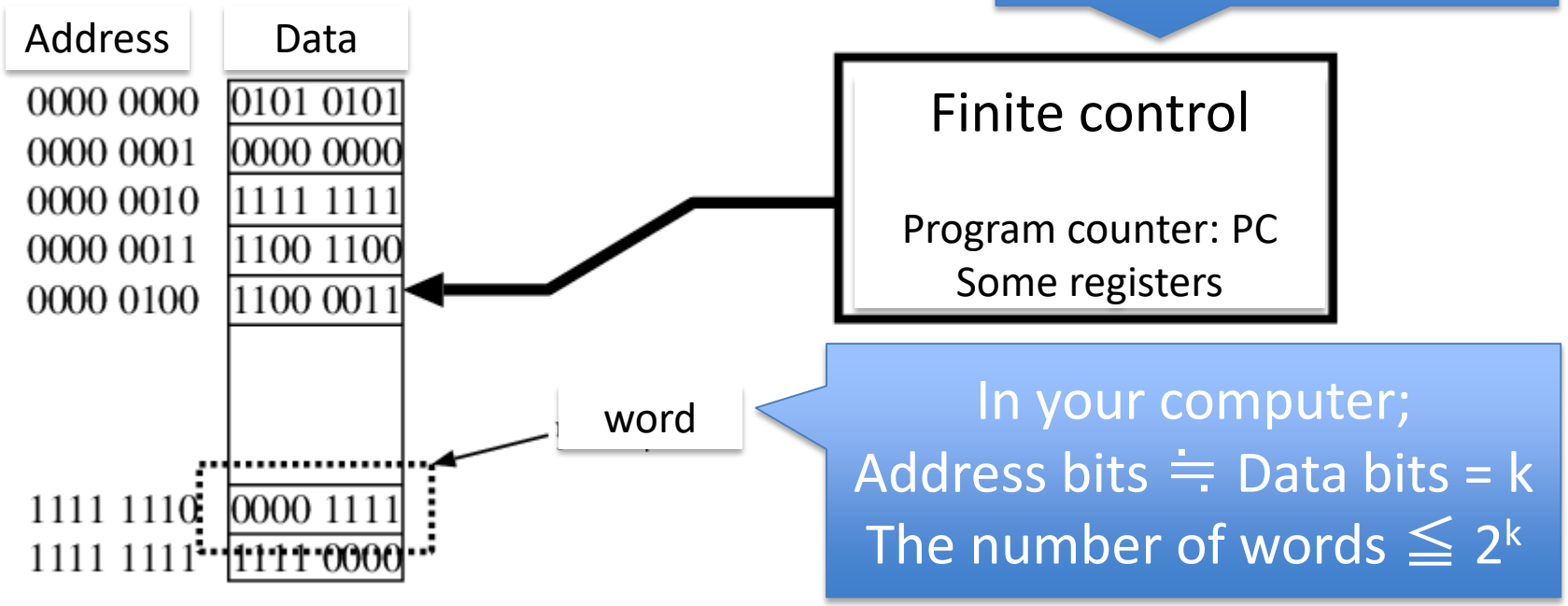



- Simple theoretical model
- Any computable problem is also solvable by a Turing machine
- It is so simple that programming is very tedious
  - No mathematical operations including +, -, ×, ÷
  - It is hard to consider “essence” of algorithms

# RAM Model

## (Random Access Mem

When we design an algorithm, we suppose memory is so huge that we have no overflow.




- 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 shows 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

- We use C language, but the other languages (C++, C#, Java, etc.) are basically similar
- We give very rough basic programming
- Output “Hello World” on display

```
#include <stdio.h> /*use printf*/

main(){
    printf(“Hello World”);
}
```

statement

Semi-colon after a statement

★ In C#, use System.Out.WriteLine instead of printf.

# Basic of C: Arithmetic operations

- Basic operations: `+`, `-`, `*`, `/`, `%`

| Exp. | 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 |
| 3%2 | Reminder by dividing 3 by 2 |

  - Except `%`, the operations can be used for `integers` (`int`, etc.) and real numbers (`float`, `double`, etc.)

# Basic of C: Notes for arithmetic ops.

- (int/int) is rounded (by cutting off)
  - Ex: `1/3 is 0, whereas 1.0/3 is 0.3333…`
- `double av = (int)sum/(int)num (Fail)`
- No comma for delimiter
  - Ex: `10,000 is not okay. Write as 10000.`
- We use () to control ordering:
  - We cannot use {} or []
  - Ex: `{(3+4)*3+4}*6` is not correct. Write as `((3+4)*3+4)*6`
- No power operation (we can use ** in some languages)

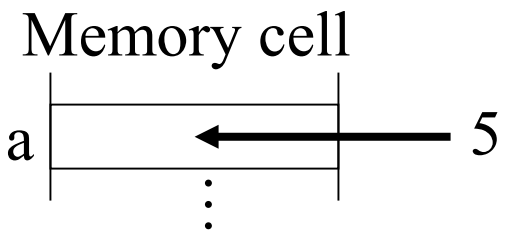
# Basic of C: Variable

- 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#`

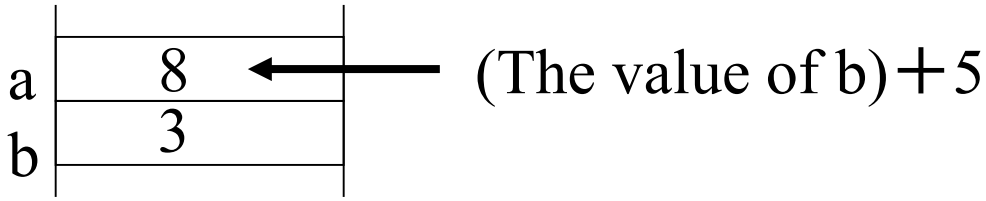
# Basic of C: Assignment statement

- a=5

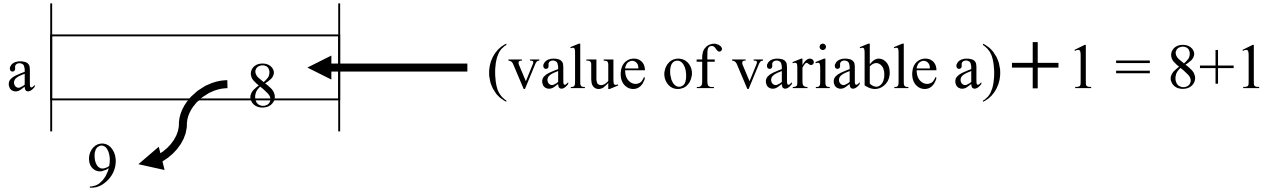



"=" is not "equal" in the sense of mathematics

  - 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. The system guesses and makes simpler, but sometimes causes bugs…

# Basic of C: Mathematical functions

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

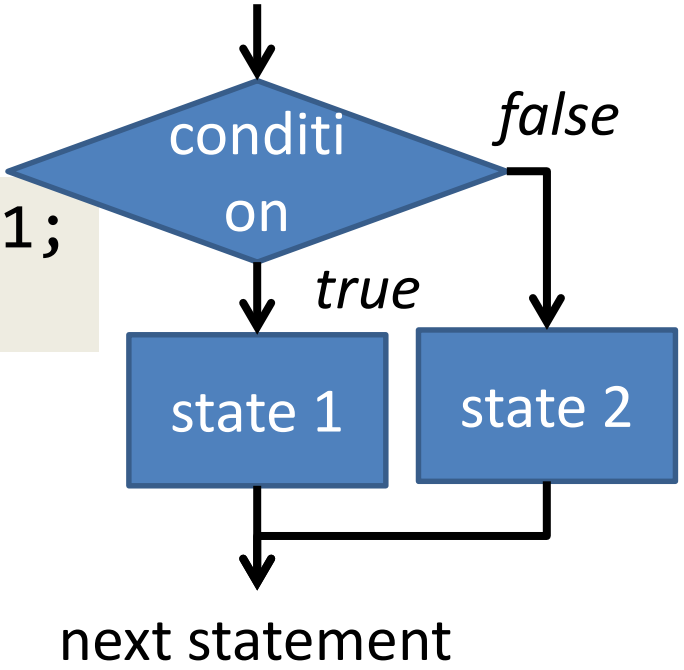
- Source code: include the following header file `#include <math.h>`
- Compile: Option -lm is required
  - gcc main.c –lm

★ Write a = Math.sqrt(b) in C#

# Basic of C: Control statements
# if statement – conditional branch (1/2)

- 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");
```

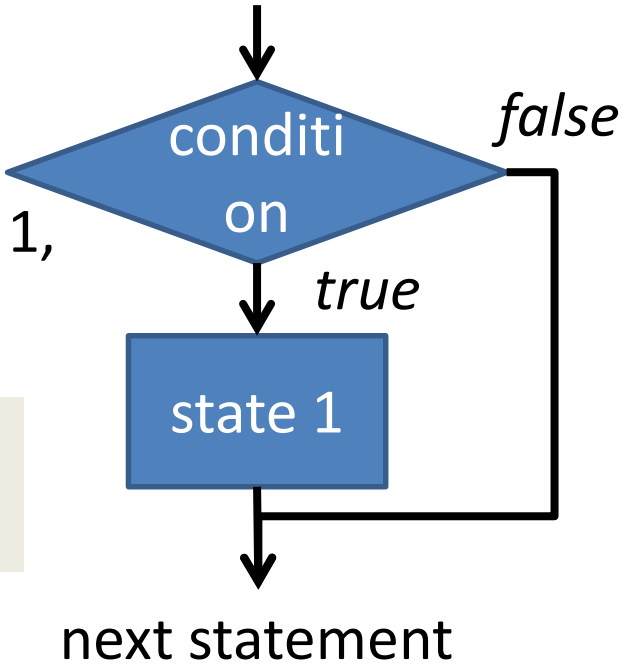
We use "==" to check equality in C.

# Basic of C: Control statements
# if statement – conditional branch (2/2)

- `else part can be omitted`

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

If condition is true, perform statement 1,
and perform nothing if it is false




```
What happens??:
if(condition) state 1; state 2;
```

```
Write as follows:
if(condition) {
  state 1;
  state 2;
}
```

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

| symbol | meaning | example | meaning of example |
|---|---|---|---|
| == | equal | n == 2 | n is equal to 2 |
| != | not equal | n != 0 | n is not equal to 0 |
| > | greater than | n > 3 | n is greater than 3 |
| >= | g.t. or equal | n >= 3 | n is g.t. or equal to 3 |
| < | less than | n < 0.01 | n is less than 0.01 |
| <= | l.t. or equal | n <= 0.01 | n is l.t. or equal to 0.01 |
| && | and | 0 < n && n <= 10 | n is greater than 0 and less than or equal to 10 |
| \|\| | or | n < 0 \|\| 0 < n | n is less than 0 or greater than 0 |
| ! | not | !(n < 0.01) | 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` ➔ `0 < x && x < 5`
  - `a==b==c` ➔ `a == b && b == c`

- Example: Check of leap year
  - Dividable by 400, or
  - Not dividable by 100 but dividable by 4

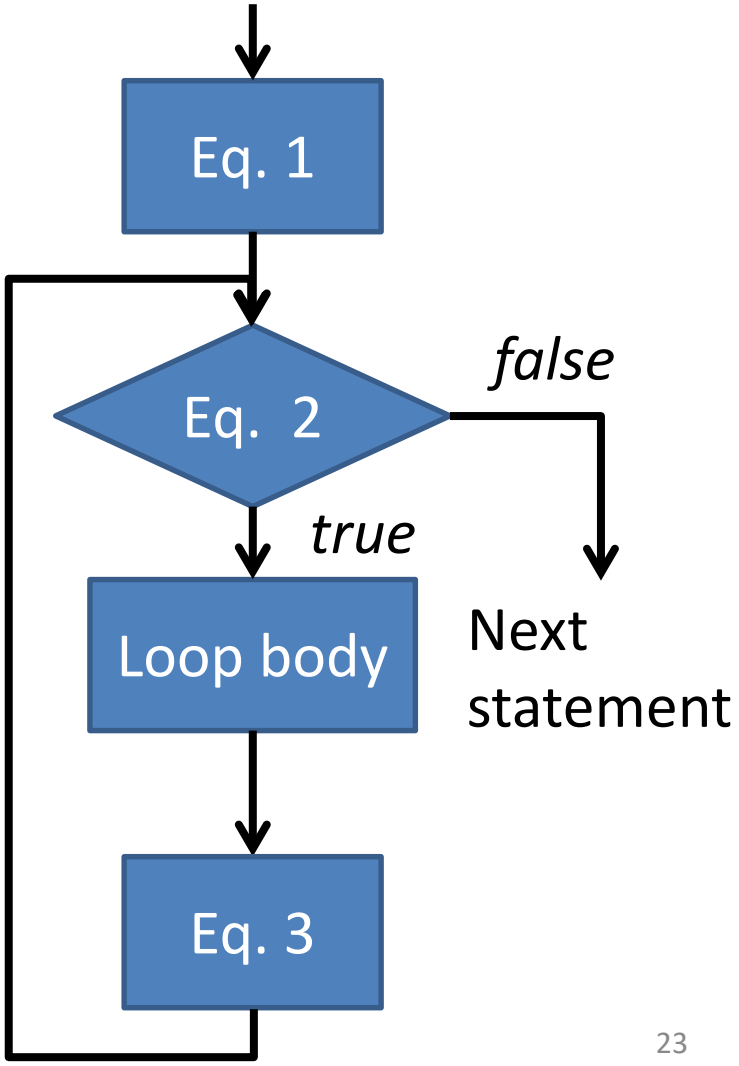
```
year%400==0 || (year%100!=0 && year%4==0)
```

# 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




At a glance, it seems to be complex,
but we have several standard patterns.

# Basic of C: Control statements for loop – repeating (2/4)

Example: Output the sum $\sum_{i=1}^{n} i$ from 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);
```

In C,
you can write i++
instead of i=i+1, and

you can write
sum+=i instead of
sum=sum+i

★ You may write as System.Console.WriteLine ("1+…+"+n+"="+sum) in C#

# Basic of C: Control statements for loop – repeating (3/4)

Example: Output the sum $\sum_{i=1}^{n} i^2$ from 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; $\sum_{i=1}^{n}(2i-1)^2 = 1^2 + 3^2 + \cdots + (2n-1)^2$

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

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

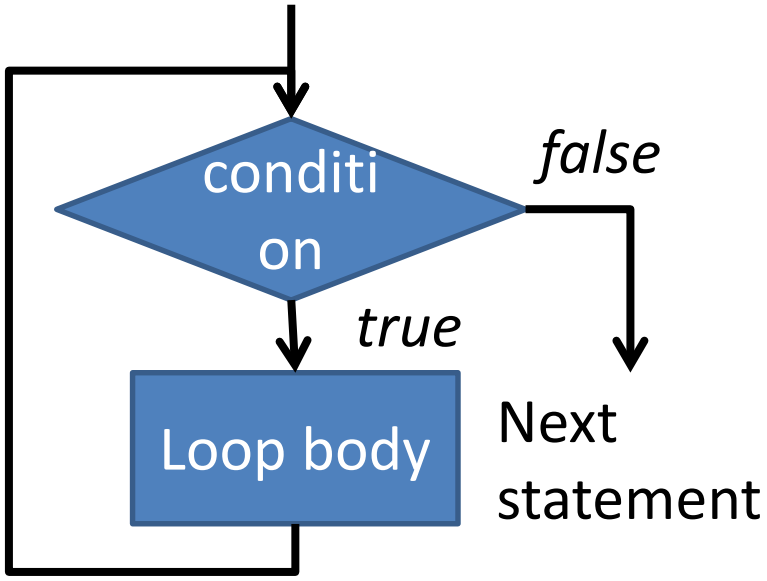
```
int i,n,sum;
n=/*initialized somehow*/;
sum=0;
for(i=1;i<=n;i=i+1){
  sum=sum+(2*i-1)*(2*i-1);
}
```

- Of course, you can do in this way.

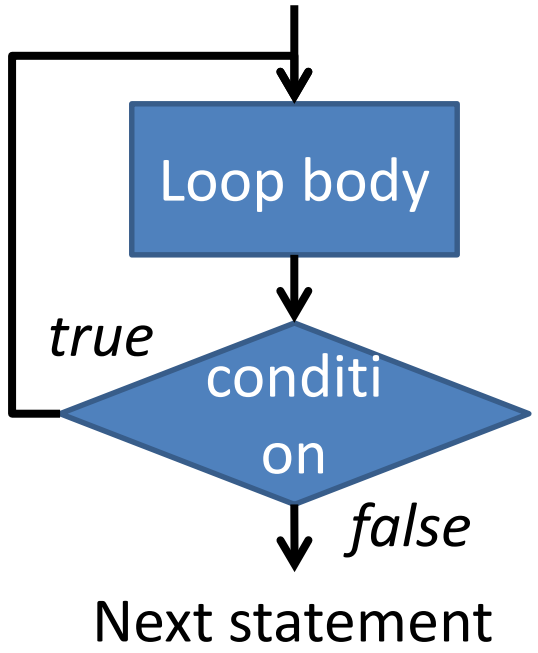
# Basic of C: Control statements while loop & do-while loop (1/2)

- 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", which is known as the oldest algorithm.

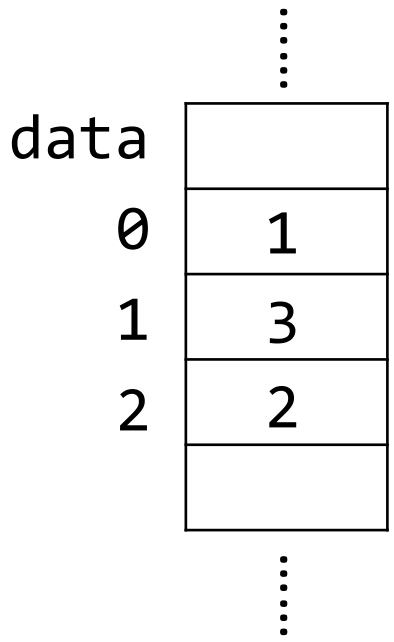
★ In C#, int[] data = new int[3];

# 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

Not only "values" in recent language.

- 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);
```

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
}
printf("maximum data = %d",max);
```

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

# Small Exercise (1)

- What does the following function compute?
  - Find the outputs of collatz(5) and collatz(7)

```
collatz(unsigned int n) {
  print(n);  // output n
  if (n == 1) return;
  if (n%2==0) collatz(n/2);
  else        collatz(3n+1);
}
```

Function calls itself recursively with different parameters

# Small Exercise (2-1)

- Definition of ExOR $\oplus$:
  - $0 \oplus 0=0$, $0 \oplus 1=1$, $1 \oplus 0=1$, $1 \oplus 1=0$
- For integers in binary system, we apply ExOR bitwise; for example,
  - $10_{10} \oplus 7_{10} = 1010_2 \oplus 111_2 = 1101_2 = 13_{10}$

"Exclusive OR" operation

1. Compute the following
   1. $8_{10} \oplus 3_{10}$
   2. $15_{10} \oplus 7_{10}$

# Small Exercise (2-2)

2. What does this function S(x,y) do?

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

Hint: Try computing
(x=8, y=3),
(x=15, y=7),
(x=1, y=128),
and so on…