Review for Exam II

JavaScript

- Building blocks
  - Variables (name, value)
  - Expressions (=, +, -, *, /, function call, if, while, for, etc.)
  - Functions (predefined, user-defined)
- Connecting to html Web-page
  - Objects, properties
    - Objects: window (status, location), document (bgColor, fgColor), image (src), link, form (text input object, button input object)
  - Web-page functions
    - alert, confirm, prompt, docoument.write, etc.
  - Event, event-driven programming
    - Event producers: link, button, etc.
    - Events and event handlers: onMouseOver, onMouseOut, onClick, etc.

Machine architecture

- Switches using transistors
- CPU
  - Control Unit - control the functioning of the other CPU components
    - IR
    - PC
  - Registers
    - Register address
  - ALU
  - Buses - Connections between registers, ALU, and memory interface
- Memory
  - Memory address
- Buses - Connection between memory and CPU
- The concept of stored program
  - Instructions are stored in memory
  - CPU's control unit loads instructions into CPU and executes them one by one

Unsolvability, nonfeasibility and halting problem

- Unsolvable
  - Halting problem
    - The existence of its solution will cause contradictory situations.
    - Therefore no solution exist
- Nonfeasible
  - Require too many steps

JavaScript

Variable

- **Variable** is a “container” to store the information you want to store.
- Variable value: The content stored in the “container”; it can change during the execution.
- Variable name: A name to refer the information in the “container”. Naming rules:
  - Variable names are **case-sensitive**.
  - They must begin with letters (a-z, A-Z) or underscore character(_).

Data types

- The types of information that can be stored in variables are called data types.
- **Numbers**
  - Integers: positive, 0, or negative.
    - Representation in JS: As in math
  - Floating-point numbers
    - Representation in JS
      - With decimal point: 314.15
      - With scientific notation: 3.1415e2
- **Booleans**: **true** or **false** (the case does matter!)
- **Strings**: Strings are delineated by single (‘) or double quotation (“) marks. (Use single quotes to type strings that contain quotation marks.)
  - E.g. **“**This is course CIS 1.0**”**
- Objects
- Null
- Undefined

Expressions

- Each JavaScript data type is associated with a specific set of predefined operators.
- Strings can be concatenated using the + operator.
  - E.g str = “first half” + “second half”
- Numbers have a predefined standard arithmetic operators +(addition), -(subtraction), *(multiplication), and / (division)
  - E.g t = 10 + 4/2 + 3 * 5
- An ***expression*** is any valid set of literals, variables, operators, and expressions that evaluates to a single value; the value can be a number, a string, or a logical value.

JavaScript Function

- A JavaScript function is an abstraction of sequence of instructions for a certain task.
- Parameters – Input of the task
- Local variables – Local memory to perform the task
- Return statement – Return the result of task back to the calling program (which will be accepted into the calling program's memory).
- Note: Some functions perform Input/Output tasks; the result of these functions are not only reflected by the return statement and memory, but also reflected in the reality (display in monitor, print in printer, input of keyboard, etc.)

Advantages of using functions

- Help minimize the amount of the detail that the programmers must keep track of. You only need to care about
  - Function name
  - Input of the function
  - Output of the function (result)
- Help minimize the length and complexity of the code.
  - The lengthy sequence of instructions for a task are organized as functions.
  - Every time the task is needed, you just need to call the function without repeating the instructions again and again.

JavaScript predefined functions

- JavaScript's predefined functions represent a collection of useful, general-purpose abstractions of tasks.
- Input/Output: prompt, confirm, alert
- Math functions: Math.sqrt

Syntax of User-defined Functions

```
function FUNCTION_NAME(PARAMETER_1, PARAMETER_2, …,
    PARAMETER_n)
{
   STATEMENT1;
   STATEMENT2;
   ….
   STATEMENTm;
   return VARIABLE_NAME;
}
```

- Note: parameters and return statement are optional; you can define a function without parameters as well as a function without return statement.

Syntax

- The **syntax of JavaScript** is a set of rules that defines how a JavaScript program will be written and interpreted
  - Rules for variable names
  - Rules for valid data types
  - Rules for valid instructions
  - More…

Important Issues

- The language instructions must be written in **lower case**
- All the instructions must **be spelled correctly and exactly**
- Parts of an instruction need to be **separated by space** and not run together
- The correct **punctuations** are required

Structure of JavaScript

- Instructions are separated by semi-colons or line-breaks

```
<html>
  <head>
    <title> An JavaScript example </title>
    <script>
      name = prompt("Please enter your name");
      document.write("Hi "+ name);
    </script>
  </head>
  <body>
  </body>
</html>
```

The Places to Put JavaScript

- Inside <head>…</head>
  - Illustration: <**head**>… <script>…</script>… </**head**>
  - The **functions** and **variables** defined in <script>…</script> inside **head** are guaranteed to be **established** before the execution of any JavaScript codes in <body>…</body>.
  - The Javascript codes inside **head** are guaranteed to be executed before any codes in <body>…</body>.
- Inside <body>…</body>
  - Illustration: <**body**>… <script>…</script>… </**body**>

About Java Script

- Interpreted high level programming language
- Purpose
  - Dynamic changes to the webpage
  - Real time changes to the webpage
- History
  - Netscape with Sun Microsystems developed it as a web programming language
  - Since Netscape Navigator 2.0
  - Since Microsoft Internet Explorer 3.0
- Characteristics of the java script
  - **Allows interactive content on webpage**
  - **Client-based: work on the browser-side not the server-side**
  - No manipulation of files and directories
  - Does not carry out graphics

JavaScript Objects

- An object is a collection of **properties** and **methods** to access and modify the properties
- An example: html **docment** in JavaScript
  - Properties
    - document.fgColor
    - document.bgColor
    - Etc.
  - Methods
    - document.write
    - Etc.

String concatenation for HTML element generation

- Italics text:
  - variable_for_text = “DYNAMIC TEXT CONTENT”;
  - document.write(“” + variable_for_text + “”);
- General scheme
  - variable_for_the_dynamic_content = …
  - document.write(“<tag>” + variable_for_the_dynamic_content + “</tag>”);
- More general
  - You can use string concatenation to produce any string.
  - If the string is a valid piece of html elements (e.g. “<img src=‘myimage.gif>’”), you can output the string for you intended html effects using **document.write(string).**

window.prompt

- Function: Displays a Prompt dialog box with a message and an input field.
- **Syntax:** [userInput]=prompt(*message*, [*inputDefault*])
- **Parameters**
  - *message* is any string or a property of an existing object; the string is displayed as the message.
  - *inputDefault* is a string, integer, or property of an existing object that represents the default value of the input field. **InputDefault is optional; it can be omitted.**
- **Return value**
  - userInput is the string which the user inputs on the dialogue box.
- **Example**
  - **document.prompt(“Please enter a year”, “2006”);**

document.write

- **Function:** Writes one or more HTML expressions to a document in the specified window.
- **Syntax :** document.write(*expression1* [,*expression2*], ...[,*expressionN*])
- **Parameters**
  - *expression1* through *expressionN* are any JavaScript expressions or the properties of existing objects.
- Example:
  - document.write(“This is a message produced by write method”);

window.alert(string)

- Function: Displays an Alert dialog box with a message and an **OK** button.
- **Syntax**
  - alert("*message*")
- **Parameters**
  - *message* is any string or a property of an existing object.
- Example: alert(“This is an alert message”);

document.bgColor

- A property of document: A string specifying the color of the document background.
- **Syntax**
  - document.bgColor
- Example: document.bgColor = “red”
  - This instruction will set the document background color to be red.

window.confirm

- Function: Displays a Confirm dialog box with the specified message and **OK** and **Cancel** buttons.
- **Syntax**
  - [userResponse]=confirm("*message*")
- **Parameters**
  - *message* is any string or a property of an existing object.
- Return value
  - userResponse, which is optional, will be true if the user press OK button, and will be false if the user press Cancel button.
- Example: confirm(“Please confirm this message”);

window.open

- Function: Opens a new web browser window.
- **Syntax**
  - [*windowVar* =][window].open("*URL*", "*windowName*", ["*windowFeatures*"])
- **Parameters**
  - *windowVar* is the name of a new window. Use this variable when referring to a window's properties, methods, and containership.
  - *URL* specifies the URL to open in the new window. See the *location* object for a description of the URL components.
  - *windowName* is the window name to use in the TARGET attribute of a FORM or <A> tag. *windowName* can contain only alphanumeric or underscore (_) characters.
- Example: window.open(http://www.gc.cuny.edu, “aNewWindow”);

window.close

- Function: Closes the specified window.
- **Syntax**
  - *windowReference*.close()
- **Parameters**
  - *windowReference* is a valid way of referring to a window, as described in the window object.
- Example: window.close()
  - Close the current window; window is a windowReference to the current active window.

window.moveBy

- Function: Moves the window by the specified horizontal and vertical offsets.
- Syntax: window.moveBy(param1, param2)
- Parameters:
  - param1: the horizontal offset in pixels.
  - param2: the vertical offset in pixels.

HTML form

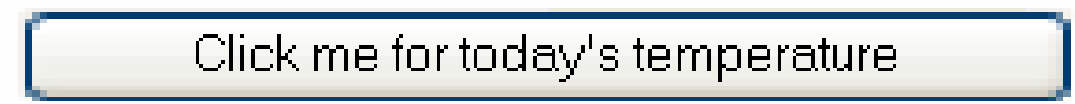
- A **form** is a grouping of buttons and other event-driven elements within a page, delimited by the tags <form name="FORM_NAME"> … </form>

```
<html>
  <head>
   </head>
   <body>
      <form name="MyForm1">
        <input type="button" value="Click Here" />
        <input type="text" name="temperature" value="110" />
      </form>
   </body>
</html>
```

Button and its event handler

<input type="button" value="BUTTON_LABEL" onClick = "JAVASCRIPT_CODE" />

- **type = "button"** specifies to the browser that this is a button element.
- value="BUTTON_LABEL" specifies the label to display on the button face; BUTTON_LABEL should be substituted by any text you what to display on the button face.
- onClick = "JAVASCRIPT_CODE" specifies the JavaScript code to be executed when the button is clicked by a user.
- E.g. <input type="button" value="Click me for today's temperature" onClick = "**alert('93° F ');**"/>



Text object

<INPUT TYPE="text" NAME="*textName*" VALUE="*textValue*“ />

- TYPE = “TEXT” specifies that this is text input element of a form.
- *NAME="textName"* specifies the name of the *Text* object. You can access this text object in JavaScript by
  - Document.formName.textName where formName is the name of a form which contains this text object.
- *VALUE="textValue"* specifies the initial value of the *Text* object. The value of the text object can be changed by user or by javascript. You can access the value of this text object in JavaScript by
  - Document.formName.textName**.value**
- E.g. <INPUT TYPE="text" NAME="temperature" value="95° F" />

HTML images with JavaScript

<IMG [NAME="*imageName*"] SRC="*Location*" />

- An image is **an object** of document's images in JavaScript.
- *NAME="imageName"* specifies the name of the *Image* object. imageName should be replaced by any name you want to name your image. You can access this image in JavaScript by
  - **docment.images**.imageName
- *SRC="Location"* specifies the URL of the image to be displayed in the document. You can access this value using the *src* property.
  - **docment.images**.imageName.src
- Change the image
  - **document.images**.imageName.src = "imageFilePath"

Events and Event Handlers

- An **event** is a user action.
- An event handler is JavaScript code which responds to the user action.
- An example: onMouseOver

```
<A HREF="#"
        onMouseOver = "doucment.bgColor = 'red'; return true"
>
Watch this link!
</A>
```

Event handling using functions

```
<html>
  <head>
  <script>
  function promptAndCalc()
    {
      tempInFhr = prompt("Please enter temperature in Fahrenheit");
      tempInFhr = parseFloat(tempInFhr);
      celius = (5/9) * (tempInFhr - 32);
      alert("Temperature in celius is " + celius);
    }
  </script>
  </head>
  <body>
      <input type="button"
             value="Click me for your celius temperature"
             onClick="promptAndCalc();"/>
   </body>
</html
```

Machine Architecture

The Architecture

- CPU
  - ALU
  - Control Unit
    - PC
    - IR
    - MM – memory address
  - Registers
  - Buses connecting ALU and registers
- Main memory
  - Memory cell
  - Address
- Buses connecting CPU and Main memory

ALU

- The **arithmetic logic unit (ALU)** is the collection of circuitry that performs actual operations on data.
- **Basic operations** include
  - Addition
  - Subtraction
  - Bit manipulation (such as shifting or combing bits)

Registers

- **Registers** are memory locations that are built into the CPU.
- Data in registers can be accessed more quickly than the data in memory (as much as 5-10 times faster).
- Limited number of registers in CPU due to the cost (commonly 16 or 32 registers).

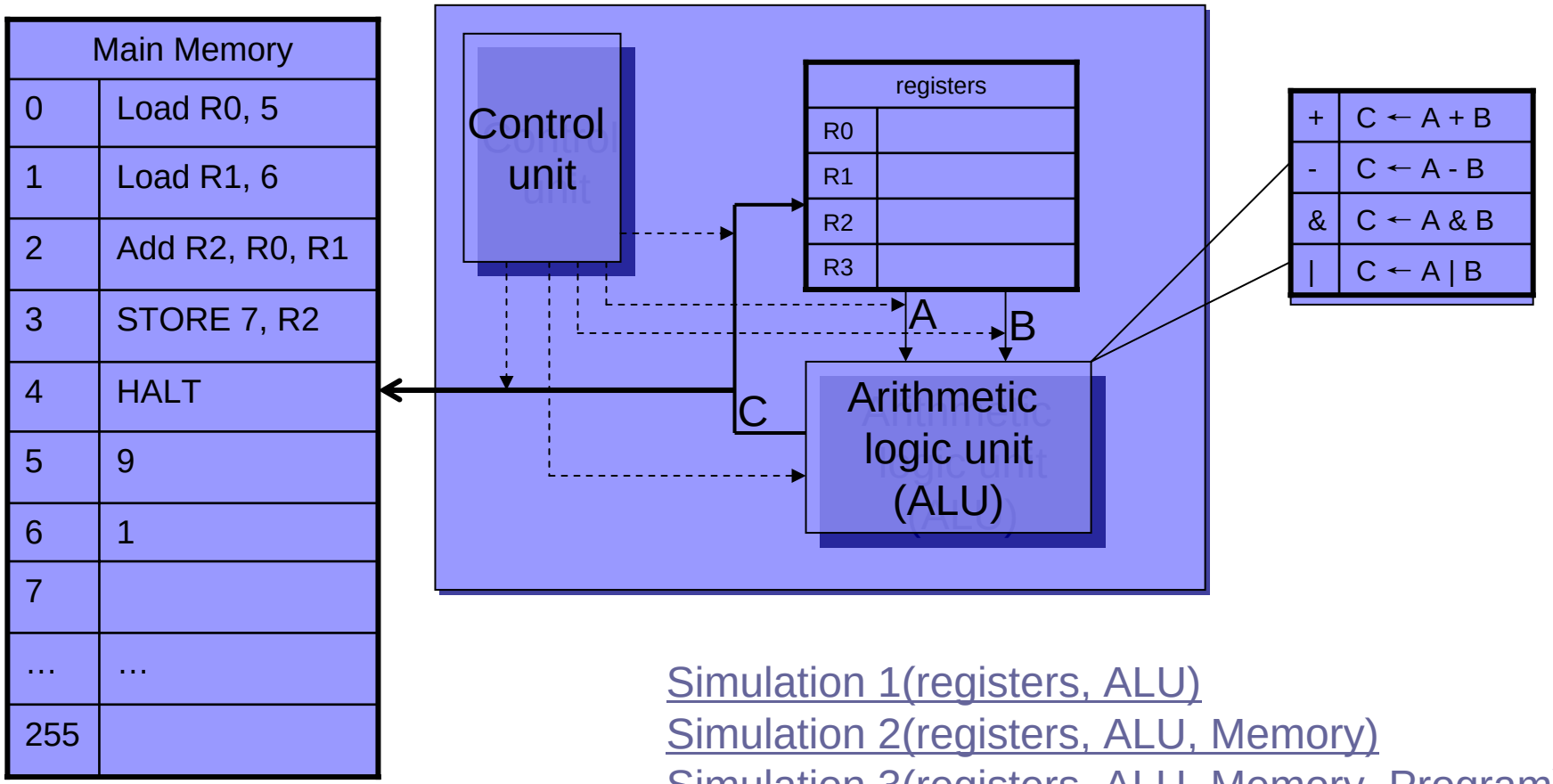
Buses

- **Bus** is a collection of **wires** which are responsible for transferring data between computer components.
- A set of buses inside CPU connect the registers to ALU.
- A bus Between CPU and memory connect the memory to CPU.

Control Unit (CU)

- The **control unit (CU)** can be thought as “the brain within the brain”, in that it oversees the various functions of the CPU.
- The control unit is a set of circuitry.
- Control unit is in charge of
  - Fetching **data** from main memory to CPU
  - Fetching **instructions** from main memory to CPU
  - Controlling the **flow of data** from **registers to the ALU** as well as from **ALU to registers**
  - Controlling the **operations of ALU**
  - Storing **data** from CPU to main memory
  - Storing **instructions** from CPU to main memory
- Basic controlling means are setting the **switches.**

CPU and Main memory



[Simulation 1(registers, ALU)]
[Simulation 2(registers, ALU, Memory)]
[Simulation 3(registers, ALU, Memory, Program)]

More Details about Control Unit

- **Program Counter (PC)** contains the memory address of next instruction to be executed.
- **Instruction register (IR)** contains the instruction that the control unit is currently executing.
- Configuration of switches for
  - ALU
  - ALU-register bus
  - Memory-CPU bus
  - Addresses of selected registers
  - Address of selected memory

Main Memory

- We can think of main memory as a large collection of **memory locations (cells)**.
- Each location is labeled by an address (binary number).
- Each location can be accessed by given its address.
- A **bus** connects main memory to CPU.
- A little bit details
  - A memory address (a sequence of 0s and 1s) will activate **a set of switches** which **select** a specified memory location.
  - The switches connect this specified memory location to **an interface** connecting to the Memory-CPU bus.

Instruction Cycle

- Fetch the instruction from main memory whose address is in the PC (program counter).
  - Store the instruction in IR (instruction register)
  - Increase the instruction address in PC by 1
- The control unit (CU) decodes the instruction and figure out the configuration of the switches to select the registers, buses, memory, etc.
- The control unit (CU) executes the instructions by activating the switches following the configuration given figured out above.
- A little bit more details: Some instruction (e.g. Jump xxx) is just intended to modify the content of PC, so as to modify the execution path of the program.

Speed of Today's Computer

- Millions or billions of instructions are being executed in one second in modern computer

The limit of computer

Computability

- A problem is **computable** if it is possible to write a computer program to solve it.
- A problem is **noncomputable** if it is impossible to write a computer program to solve it. Such a problem is also called **unsolvable**.

Nonfeasibility and Running Time

- **Non-feasible** – a computable problem that takes too long to solve (with the fastest algorithm).

Loop

- A repetition of a sequence of instructions
  - Loop conditional test: determine whether to execute the loop body
  - Loop body: the sequence of instructions to be repeated
- Infinite loop
  - Repeat the execution of the loop body forever
  - Loop condition test is always true

Loop at the machine level

Execution sequence: 0, 1, (2,3,4),(2,3,4)..., (2,3,4), 5

0: x = 0
1: y = 0
2: x = x + 2
3: y = y + 1
4: if (y – 10 < 0) then
jump to 2
5: halt

The task of above program is to compute 2*10 with using only the operations of additions.

- Program counter (PC) stores the **address of the instruction to be executed**.
- After an instruction has been executed, **the address in PC is increased by 1**.
- With the above settings, instructions are executed sequentially according to **their appearance ordering in the memory.**
- **Jump**: To **modify the address stored in PC**, so that change the instruction execution ordering
- **Loop:** Jump to **the address of previous executed instructions**, so that **a sequence of instructions will be executed again and again**

The Halting Problem

- Halting problem – given a **computer program** and **an input to the program**, will the **given program HALT** on the given input?
- Can we write a program to solve the halting problem? Namely, can we write a program **X** to tell whether a computer program will terminate given input **Y**?
  - **NO, WE CAN'T!**
- It is a problem **can not be solved** by computer! – It is a so-called **unsolvable problem**, or **non-computable** problem!

Why Halting Problem Unsolvable?

- Assume we can solve the halting problem, then we can write a function **will_halt**(program_X, input_Y), which
  - returns true, if program_X(input_Y) can terminate
  - return false, if program_X(input_Y) will run forever
- Some reminders about the inputs of **will_halt**
  - **program_X** is a sequence of instructions sitting in some area of the memory, essentially it is just **a sequence of 0s and 1s** just like the other data in memory**.**
  - **Input_Y** is the input data which program_X will work on**.** Since program_X is also data in memory, the setting allow the user use program_X as input to call program_X, namely the user can **set input_Y = program_X.**
- Can we have the function **will_halt**?
  - **NO!**

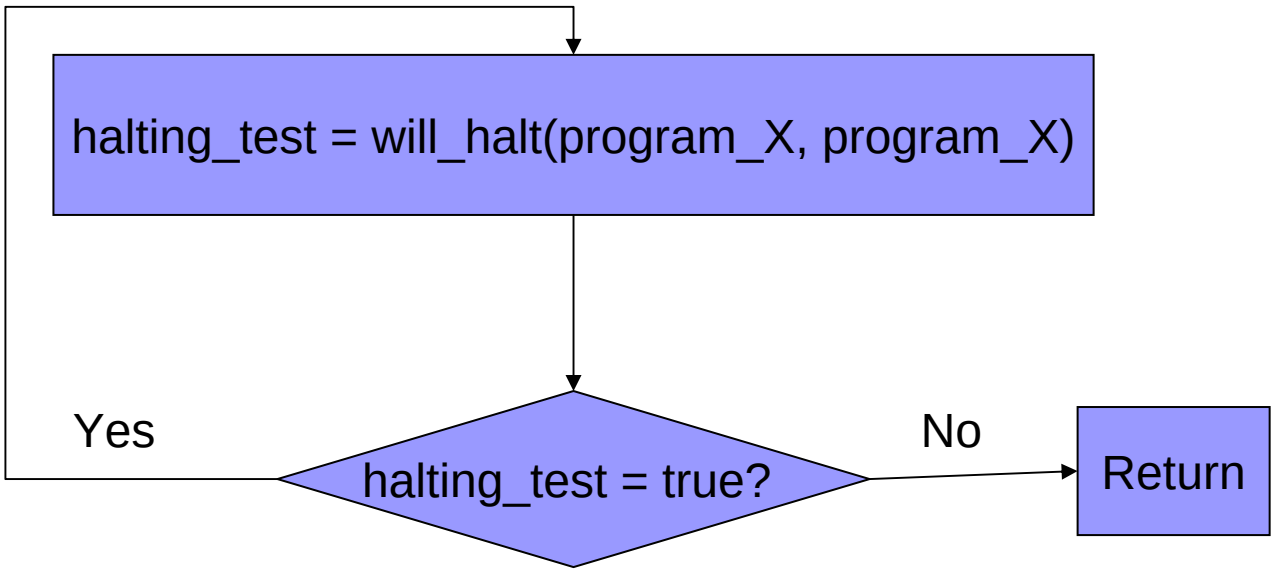
Solutions to halting problem cause contradictions!

```
function self_check(program_X)
{
   halting_test = true;
   while (halting_test == true)
   {
       halting_test = will_halt(program_X, program_X);
   }
}
```

- Contraction arises when calling **self_check(self_check) (namely, prograx_X=self_chck):**
  - If **self_check(self_check) halts**, **then** halting_test = will_halt(self_check, self_check) will be true forever, which means that the **self_check(self_check) will run forever**. **A contradiction!**
  - If self_check(self_check) doesn't halt, then halting_test = will_halt(self_check, self_check), which means that the **self_check(self_check) will terminate. Another contradiction!**
- **This means a paradox**
  - **If we can have a computer program to check whether another program halt or not, then such a halting checking program will always cause contradictory situations!**
  - **But contradictory situations can not happen in the real world, therefore we can not have a computer program to check whether another computer program halt or not!**

Diagram of contradictory situations caused by the halting problem

function **self_check**(program_X)



- self_check(self_check) halts -> (“Yes” path) continue the loop -> means run forever! **(contradiction !)**
- self_check(self_check) runs forever -> (“No” path) stop and return -> terminate!! **(contradiction!)**

Turing-Church Thesis

- *Thesis: "Every 'function which would naturally be regarded as computable' can be computed by a Turing machine."* (Turing machine is a model of computer.)
- The **Church–Turing thesis** (also known as the **Church's thesis**, **Church's conjecture** and **Turing's thesis**) is a hypothesis about the **nature of computers**, such as a **digital computer** or **a human** with a pencil and paper following **a set of rules**.

Examples of feasible problems

- Adding to two integer numbers
  - several computer instructions
- Searching a list of n items
  - Need to do about **n** operations (to check the n items in the list one by one)
- Sorting a list of n items, one of many solutions:
  - Search for the smallest element, put it in the beginning (cost about **n** operations)
  - Do this n times for each i^{th} smallest element
  - Overall: about n^2 operations

Examples of non-feasible problems

- List all the possible score permutation of a class
  - Given the class size n, score are between 0-100
  - Need to do at least 100^n operations to output the result (assuming 1 operation for 1 permutation for simplicity)
  - For a class of 30 students, we at least need to do $\mathbf{10^{60}}$ operations
    - Modern computer can perform about 10^9 instructions in a second
    - There is roughly $3.15 * 10^7$ seconds per year
    - $\mathbf{10^{60}} / (3.15 * 10^7 * 10^9) > \mathbf{3 * 10^{10}}$ years!!!
- Consider all the possible moves and outcome of chess playing
- Consider every possible seating arrangement for the class
- In general, any solution that considers every possible subset of a set requires exponential steps in term of the size of the set!

A list of steps in term of input size

Given a problem's input of size n

- Requiring n (linear) operations is feasible
- Requiring n^2 (quadratic) operations is feasible
- Requiring n^c (polynomial) operations is feasible, where c is constant in spite of n
- In general, requiring **less than n^c** operations is **feasible**
- Requiring **c^n (exponential: 2^n, 10^n,…) operations** is **non-feasible** when n becomes large, because the requiring amount of operations increases rapidly as n increases.

Analysis of algorithms

- Analysis of algorithms – analyzing the running times of algorithms (or programs). This field of computer science studies algorithms and their efficiency, in terms of the amount of work (and space) needed to execute an algorithm.

GOOD LUCK! ☺