Computer Programming II

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Floating Point

Floating-Point Format

$$\pm f.fff \times 10^{\pm e}$$

where:

$\pm$

is the sign (plus or minus).

f.fff

is the 4 digit fraction.

$\pm e$

is the single -digit exponent with sign.

Floating-Point Format

- Example

Notation	Number
+1.000E+0	1.0
+3.300E+4	33000.0
-8.223E-3	-0.008223
+0.000E+0	0.0

- Guard digit
  - an extra digit added to the end of our fraction during computation, in order to minimize errors

Floating Addition/Subtraction

- Example: 2.0 + 0.3

1. Start with the numbers:
2. `+2.000E+0        The number is 2.0.`

   `+3.000E-1 The number is 0.3.`

Floating Addition/Subtraction

- Example: 2.0 + 0.3

1. Start with the numbers:
2. `+2.000E+0` The number is 2.0.
   `+3.000E-1` The number is 0.3.

3. Add guard digits to both numbers:
4. `+2.0000E+0` The number is 2.0.
   `+3.0000E-1` The number is 0.3.

Floating Addition/Subtraction

- Example: 2.0 + 0.3

1. Start with the numbers:
2. `+2.000E+0        The number is 2.0.`
   `+3.000E-1 The number is 0.3.`

3. Add guard digits to both numbers:
4. `+2.0000E+0       The number is 2.0.`
   `+3.0000E-1       The number is 0.3.`

5. Shift the number with the smallest exponent to the right one digit, and then increment its exponent. Continue until the exponents of the two numbers match:
6. `+2.0000E+0       The number is 2.0.`
   `+0.3000E-0       The number is 0.3.`

Floating Addition/Subtraction

7. Add the two fractions. The result has the same exponent as the two numbers:

```
8.   +2.0000E+0      The number is 2.0.
9.   +0.3000E-0      The number is 0.3.
```

Floating Addition/Subtraction

7. Add the two fractions. The result has the same exponent as the two numbers:

```
8.   +2.0000E+0      The number is 2.0.
9.   +0.3000E-0      The number is 0.3.
```

- Normalization: a number like **`+0.1234E+0`** would be normalized to **`+1.2340E-1`**
- if the guard digit is greater than or equal to 5, round the next digit up; otherwise, truncate the number

Floating Addition/Subtraction

7. Add the two fractions. The result has the same exponent as the two numbers:

```
8.   +2.0000E+0      The number is 2.0.
9.   +0.3000E-0      The number is 0.3.
```

- Normalization: a number like **`+0.1234E+0`** would be normalized to **`+1.2340E-1`**
- if the guard digit is greater than or equal to 5, round the next digit up; otherwise, truncate the number

```
13.  +2.3000E+0     Round the last digit.
14.  ________________________________________
   +2.300E+0 The result is 2.3.
```

Multiplication

- Example: 0.12 * 11.0

1. Add the guard digit:
2. `+1.2000E-1      The number is 0.12.`

   `+1.1000E+1      The number is 11.0.`

3. Multiply the two fractions and add the exponents, (1.2 x 1.1 = 1.32) (-1 + 1 = 0):
4. `+1.2000E-1      The number is 0.12.`
5. `+1.1000E+1      The number is 11.0.`
6. ______________________________________

   `+1.3200E+0      The result is 1.32.`

7. Normalize the result.

   If the guard digit is greater than or equal to 5, round the next digit up. Otherwise, truncate the number:

   `+1.3200E+0      The number is 1.32.`

Division

- Example: 100.0 divided by 30.0

1. Add the guard digit:

```
2.   +1.0000E+2      The number is 100.0.
    +3.0000E+1       The number is 30.0.
```

3. Divide the fractions and subtract the exponents:

```
4.   +1.0000E+2      The number is 100.0.
5.   +3.0000E+1      The number is 30.0.
6.   ____________________________________
    +0.3333E+1       The result is 3.333.
```

7. Normalize the result:

```
    +3.3330E+0       The result is 3.333.
```

8. If the guard digit is greater than or equal to 5, round the next digit up. Otherwise, truncate the number:

```
    +3.333E+0 The result is 3.333.
```

Overflow and Underflow

- Overflow
  - number is too big
  - **`9.000E+9 x 9.000E+9`**
  - **`8.1 x 10^+19`**
  - IEEE floating-point standard: +Infinity
- Underflow
  - number is too small
  - **`1.000E-9 x 1.000E-9`**
  - **`1.0 x 10^-18`**
  - **`-18`** is too small ==> underflow

Roundoff Error

- 1 + 1 = 2
  - But, why 1/3 + 1/3 does not equal 2/3?
- 2/3 as floating-point is 6.667E-1
- 1/3 as floating-point is 3.333E-1
  - 2/3 should be 6.666E-1
- Floating-point arithmetic should never be used for money.
  - The more calculations you do with floating-point arithmetic, the bigger roundoff error

Accuracy

- How many digits of the fraction are accurate?
  - The accuracy depends on the calculation
  - Like subtracting two numbers that are close to each other, generate inexact results

```
1 - 1/3 - 1/3 - 1/3
1.000E+0
- 3.333E-1
- 3.333E-1
- 3.333E-1
```

or:

```
1.000E+0
- 0.333E+0
- 0.333E+0
- 0.333E+0
______________________
0.0010E+0 or 1.000E-3
```

Minimizing Roundoff Error

- Many techniques for minimizing roundoff error
  - use **double** instead of **float**
  - **twice** the **accuracy**
  - But roundoff errors still can creep in
- Computer is not as accurate as your expectation!!

Determining Accuracy

- Simple method of determining how accurate your floating point is
  - to add 1.0+0.1, 1.0+0.01, 1.0+0.001 ...
  - the result may vary across different computers

Floating Number

- Example: float.c

```c
    float number1, number2;
    float result;               /* result of calculation */
    int   counter;              /* loop counter and accuracy check */

    number1 = 1.0;
    number2 = 1.0;
    counter = 0;

    while (number1 + number2 != number1) {
        ++counter;
        number2 = number2 / 10.0;
    }

    printf("%2d digits accuracy in calculations\n", counter);
```

```c
    number2 = 1.0;
    counter = 0;

    while (1) {
        result = number1 + number2;
        if (result == number1)
            break;
        ++counter;
        number2 = number2 / 10.0;
    }
    printf("%2d digits accuracy in storage\n", counter);
```

Floating Number

- Example: float.c

```c
    float number1, number2;
    float result;               /* result of calculation */
    int   counter;              /* loop counter and accuracy check */

    number1 = 1.0;
    number2 = 1.0;
    counter = 0;

    while (number1 + number2 != number1) {
        ++counter;
        number2 = number2 / 10.0;
    }

    printf("%2d digits accuracy in calculations\n", counter);
```

```
 8 digits accuracy in calculations
 8 digits accuracy in storage
```

```c
    number2 = 1.0;
    counter = 0;

    while (1) {
        result = number1 + number2;
        if (result == number1)
            break;
        ++counter;
        number2 = number2 / 10.0;
    }
    printf("%2d digits accuracy in storage\n", counter);
```

Further Information

- 使用浮點數最最基本的觀念 - 冼鏡光
- Notes
  - Avoid using subtraction for two operands with almost the same number
  - Watch out overflow and underflow
  - Beware of roundoff, especially for two number with extremely big difference

Problem Solving

Cycle Detection

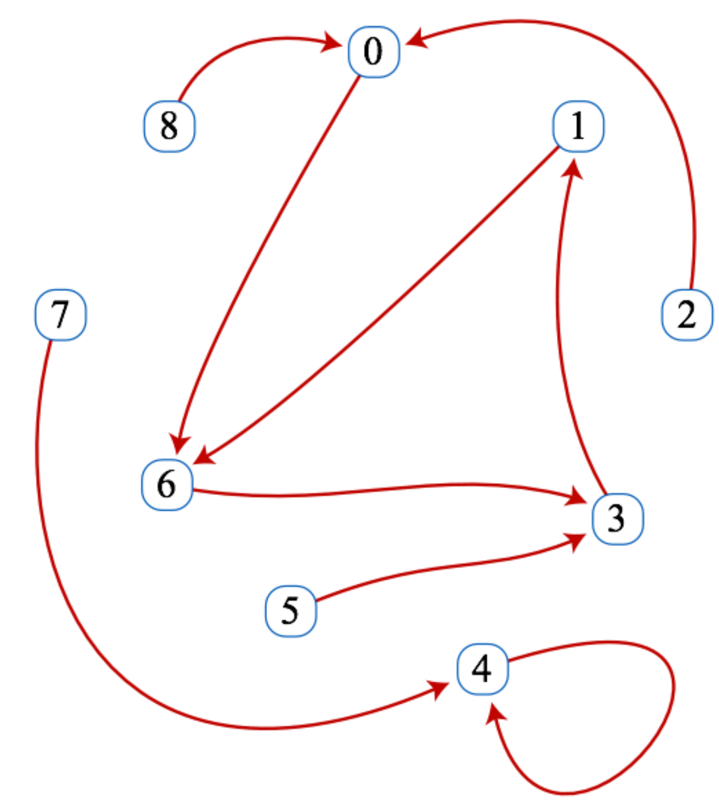
- In computer science, cycle detection is the algorithmic problem of finding a cycle in a sequence of iterated function values
- Example
  - 2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...
  - The cycle to be detected is the repeating subsequence of values 6, 3, 1 in this sequence

Cycle Detection

Note that *f* must be a function; that is, 1-to-many mapping is invalid!!

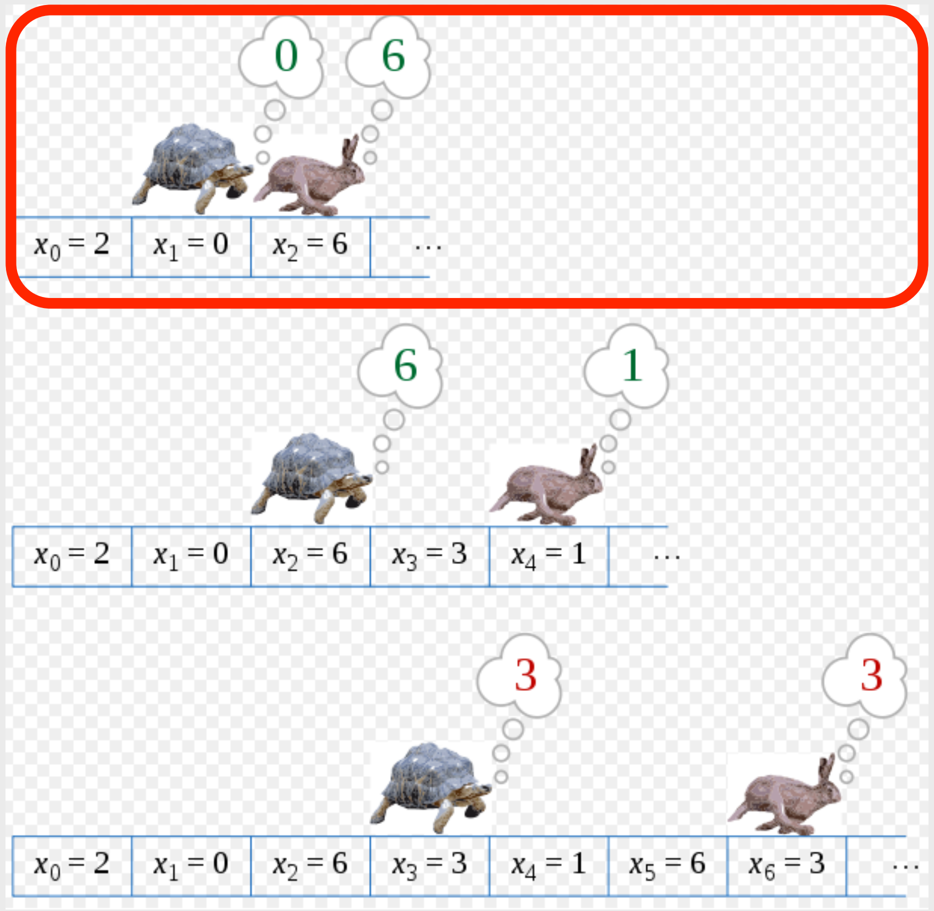
- Problem Definition
  - Let *S* be a finite set, *f* be a function from S to itself, and x_0 be a element of *S*. For any $i > 0$, let $x_i = f(x_{i-1})$. Let μ be the smallest index such that value x_μ reappears infinitely within the sequence of value x_i, and let λ (the loop length) be the smallest positive integer such that $x_\mu = x_{\lambda+\mu}$
  - The cycle detection problem is the task of finding λ and μ

x	$f(x)$
0	6
1	6
2	0
3	1
4	4
5	3
6	3
7	4
8	0



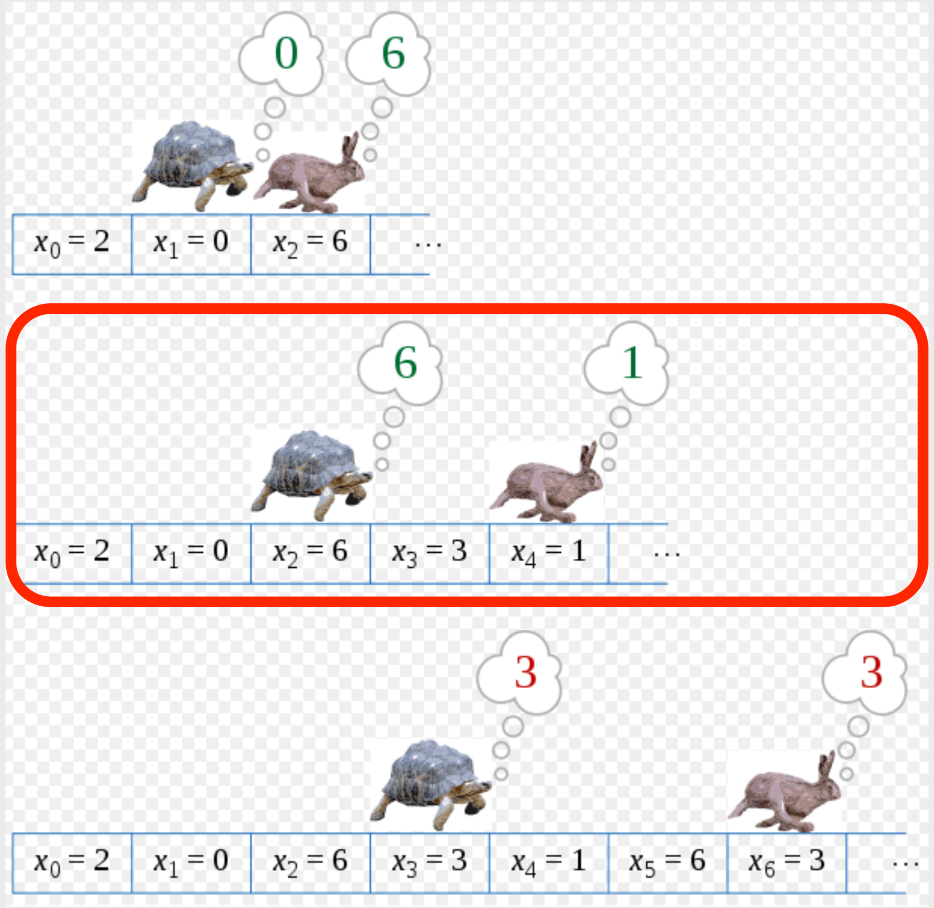
Cycle Detection

- Tortoise and Hare (Floyd's cycle-finding algorithm)
  - a pointer algorithm that uses only two pointers, which move through the sequence at different speeds



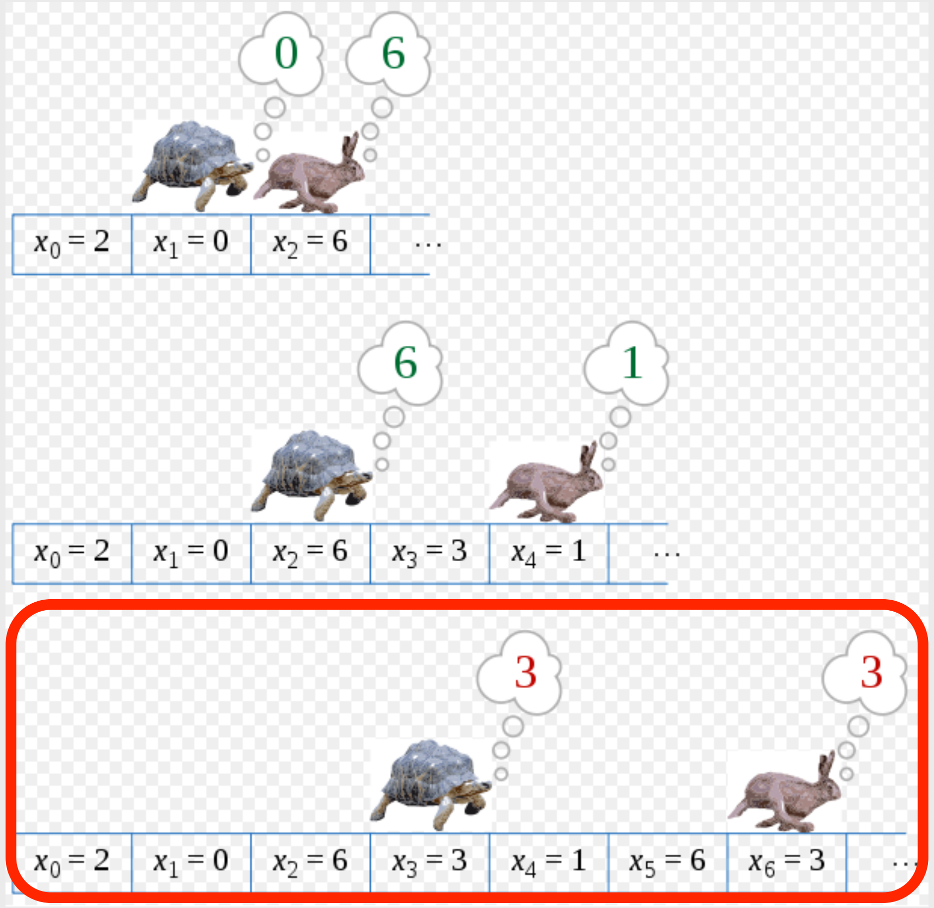
Cycle Detection

- Tortoise and Hare (Floyd's cycle-finding algorithm)
  - a pointer algorithm that uses only two pointers, which move through the sequence at different speeds



Cycle Detection

- Tortoise and Hare (Floyd's cycle-finding algorithm)
  - a pointer algorithm that uses only two pointers, which move through the sequence at different speeds



Cycle Detection

```python
def floyd(f, x0):
    # The main phase of the algorithm, finding a repetition x_mu = x_2mu
    # The hare moves twice as quickly as the tortoise
    tortoise = f(x0) # f(x0) is the element/node next to x0.
    hare = f(f(x0))
    while tortoise != hare:
        tortoise = f(tortoise)
        hare = f(f(hare))

    # at this point the start of the loop is equi-distant from current tortoise
    # position and x0, so hare (set to tortoise-current position) moving in
    # circle and tortoise (set to x0 ) moving towards circle, will intersect at
    # the beginning of the circle.

    # Find the position of the first repetition of length mu
    # The hare and tortoise move at the same speeds
    mu = 0
    hare = tortoise
    tortoise = x0
    while tortoise != hare:
        tortoise = f(tortoise)
        hare = f(hare)
        mu += 1

    # Find the length of the shortest cycle starting from x_mu
    # The hare moves while the tortoise stays still
    lam = 1
    hare = f(tortoise)
    while tortoise != hare:
        hare = f(hare)
        lam += 1

    return lam, mu
```

Cycle Detection

```python
def floyd(f, x0):
    # The main phase of the algorithm, finding a repetition x_mu = x_2mu
    # The hare moves twice as quickly as the tortoise
    tortoise = f(x0) # f(x0) is the element/node next to x0.
    hare = f(f(x0))
    while tortoise != hare:
        tortoise = f(tortoise)
        hare = f(f(hare))

    # at this point the start of the loop is equi-distant from current tortoise
    # position and x0, so hare (set to tortoise-current position) moving in
    # circle and tortoise (set to x0 ) moving towards circle, will intersect at
    # the beginning of the circle.

    # Find the position of the first repetition of length mu
    # The hare and tortoise move at the same speeds
    mu = 0
    hare = tortoise
    tortoise = x0
    while tortoise != hare:
        tortoise = f(tortoise)
        hare = f(hare)
        mu += 1

    # Find the length of the shortest cycle starting from x_mu
    # The hare moves while the tortoise stays still
    lam = 1
    hare = f(tortoise)
    while tortoise != hare:
        hare = f(hare)
        lam += 1

    return lam, mu
```

Cycle Detection

```python
def floyd(f, x0):
    # The main phase of the algorithm, finding a repetition x_mu = x_2mu
    # The hare moves twice as quickly as the tortoise
    tortoise = f(x0) # f(x0) is the element/node next to x0.
    hare = f(f(x0))
    while tortoise != hare:
        tortoise = f(tortoise)
        hare = f(f(hare))

    # at this point the start of the loop is equi-distant from current tortoise
    # position and x0, so hare (set to tortoise-current position) moving in
    # circle and tortoise (set to x0 ) moving towards circle, will intersect at
    # the beginning of the circle.

    # Find the position of the first repetition of length mu
    # The hare and tortoise move at the same speeds
    mu = 0
    hare = tortoise
    tortoise = x0
    while tortoise != hare:
        tortoise = f(tortoise)
        hare = f(hare)
        mu += 1

    # Find the length of the shortest cycle starting from x_mu
    # The hare moves while the tortoise stays still
    lam = 1
    hare = f(tortoise)
    while tortoise != hare:
        hare = f(hare)
        lam += 1

    return lam, mu
```

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

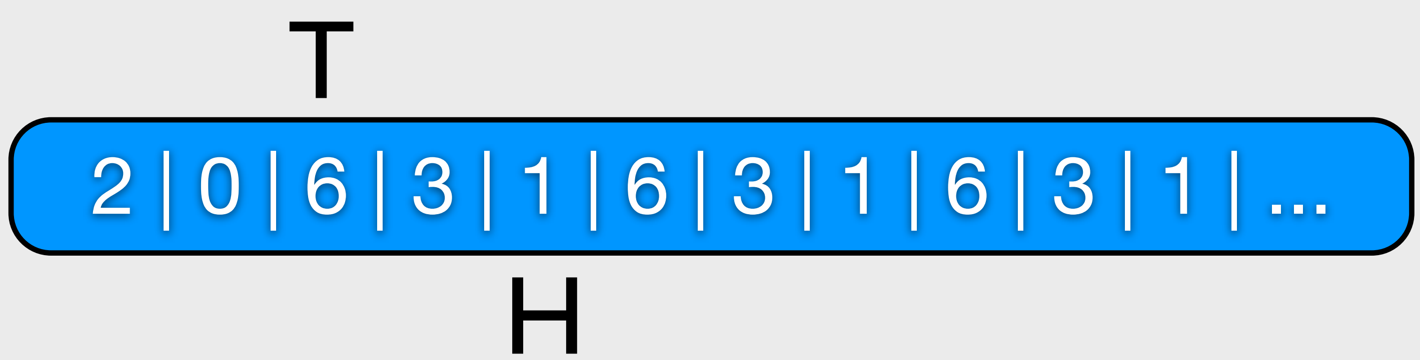
T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}



Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

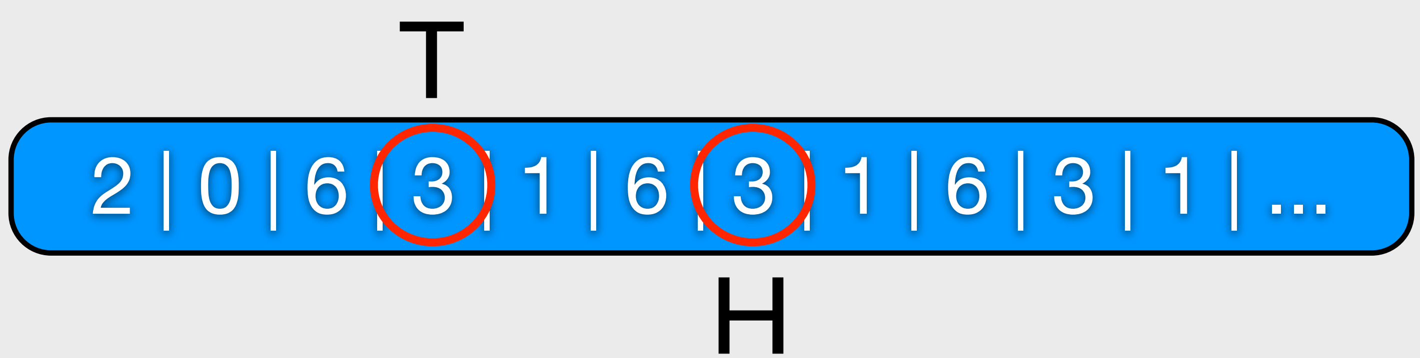
T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

Cycle Detection

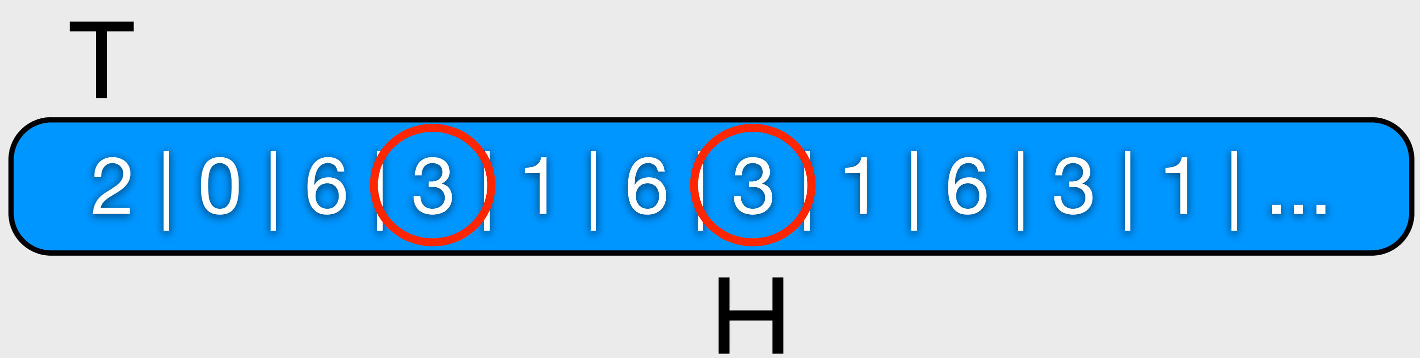
- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}



mu = 0

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}



mu = 0

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = 0

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

mu = 0

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = 0

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = I

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = l

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

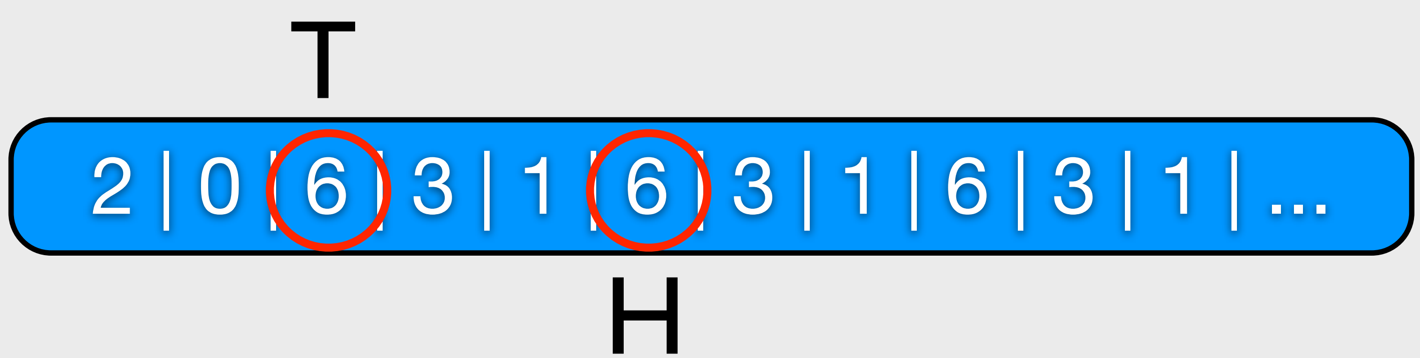
2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = 2

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}



mu = 2

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = 2

lambda = 1

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

mu = 2

lambda = 1

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = 2

lambda = 1

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = 2

lambda = 1

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = 2

lambda = 2

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = 2

lambda = 2

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

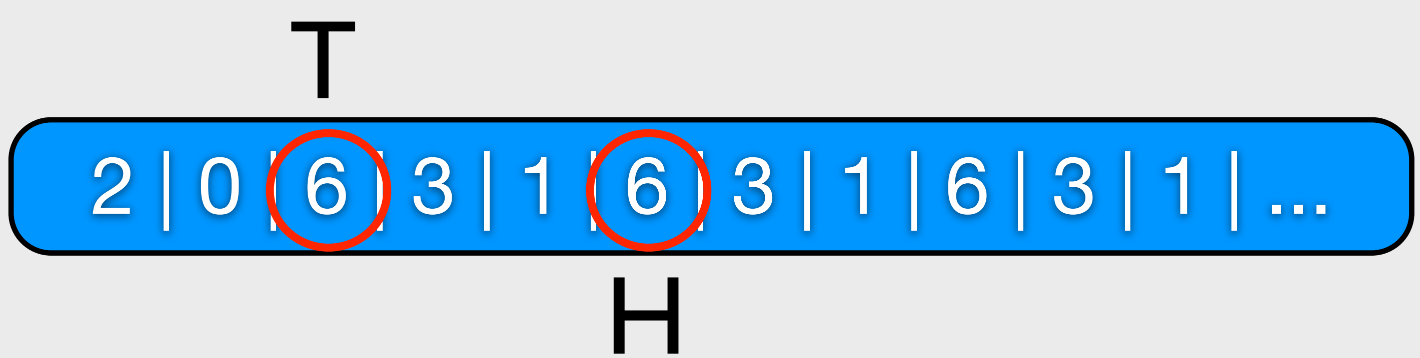
H

mu = 2

lambda = 3

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}



mu = 2

lambda = 3

Cycle Detection

- Example: {2, 0, 6, 3, 1, 6, 3, 1, 6, 3, 1, ...}

T Terminate

2 | 0 | 6 | 3 | 1 | 6 | 3 | 1 | 6 | 3 | 1 | ...

H

mu = 2

lambda = 3

Cycle Detection

- Why the Tortoise and Hare algorithm works?
- What is the time complexity?
- Further information

Cycle Detection

- Why the Tortoise and Hare algorithm works?
  - the tortoise and the hare will meet when they are $n\lambda$ apart, where λ is the loop length
  - So, if we move both one step at a time, from the tortoise's position and from the start of the sequence, we know that they will meet as soon as both are in the loop, since they are $n\lambda$, a multiple of the loop length, apart
  - One of them is already in the loop, so we just move the other one in single step until it enters the loop, keeping the other $n\lambda$ away from it at all times

Cycle Detection

- What is the time complexity?
  - Note that this code only accesses the sequence by storing and copying pointers, function evaluations, and equality tests; therefore, it qualifies as a pointer algorithm.
  - The algorithm uses $O(\lambda+\mu)$ operations of these types, and O(1) storage space.

Introduction to Scripting Languages: Bash, Perl and Python

Scripting Language

- Scripting Language
  - a programming language that allows control of one or more applications
  - batch languages or job control languages
- Difference between the core code of the application
  - written in a different language
  - interpreted from source code or bytecode

Types of Scripting Languages

- Job control languages and shells
  - Shell scripts
- GUI scripting
  - with the advent of GUI, a specialized language emerged for controlling a computer
  - such languages are called macros
- Application-specific languages
  - domain-specific programming language specialized to a single application

Types of Scripting Languages

- Web browsers
  - client-side scripting
  - JavaScript, VBScript, AJAX (XML + JavaScript)
- Text processing languages
  - one of the oldest uses of scripting languages
  - awk, sed, grep, ...
  - regular expression
- General-purpose dynamic languages
  - Dynamic programming language
  - Perl, Python, Ruby
- Extension/embeddable languages
  - ActionScript (Adobe Flash), MEL (Maya 3D), ...

Bash

Bash (Unix Shell)

- There are two main types of shells:
  - Graphical User Interface (GUI)
  - Command Line Interface (CLI).
- Bash is used by many Linux distributions as the default CLI shell.
- Bash can be used not only as a user interface to the operating system, but also as a programming environment.
- Bash is an acronym for Bourne Again SHell, named after Steve Bourne's shell (released for UNIX in 1979).

Why Bash?

- While there are other shells available, Bash has a number of distinct advantages:
  - Command Line Editing (Go back and fix typos, utilize history)
  - Tab Completion Job Control (Start, stop, pause, and background jobs)
  - Customization (For advanced users)
  - Bash is Free and Open Source Software, distributed under the GPL

Bash Basics

Type **exit/logout** or **ctrld**	exit
Type **clear** or **ctrll** (L)	clear the screen
Ctrl-c	stop current command
Ctrl-	stop current command (more forceful than ctrlc)
Ctrl-s	pause output to the screen
Ctrl-q	restart output to the screen
Ctrl-u	erase current command line
Tab	auto complete current command or filename

Simple Clean-Up Example

- Assume that we need to clean up a directory every time before compiling a program

```
$ rm –rf *.o
$ rm –rf *.bak
$ rm –rf *.exe
```

Shell scripts

- List of command, executed in order
  - `#!`: tells the CPU what shell to use to execute script
  - The shell name is the shell that will execute this script.
    - E.g., `#!/bin/bash`
- If no shell is specified in the script file, the default is chosen to be the executing shell.

The First Bash Script

```
$ mkdir ~/scripts
$ cd scripts
$ vi hello.sh
```

- Write programs using vi
- So fire up a text editor; for example:
- Type the following inside it:

```
#!/bin/bash
# This is a commented line, will not be executed
# This is my first script "Hello World"
echo "Hello World"
```

- Make the script executable:

```
$chmod u+x hello.sh
$ls –l
-rwxr--r-- hello.sh
```

The First Bash Script

- To execute the program:

```
$ hello.sh
-bash: hello.sh: command not found
```

- $PATH environment variable holds the location where all commands are stored

```
$ echo $PATH
/usr/bin:/bin:/usr/sbin
```

- We must specify the path of hello.sh

```
$ /cchome/arodrigu1/scripts/hello.sh
$ ./hello.sh
Hello World
```

Back to the Clean-Up Example

- We can put all those commands into a shell script, called mycleanDir.sh.

```
$ vi mycleanDir.sh
#!/bin/bash
rm –rf *.o
rm –rf *.bak
rm –rf *.exe
echo "Deleted files with suffix blastout, stderr, stdout, tmp"
```

Make it executable and run!

Variables

- There are two types of variables
  - Environmental variables
  - Local variables

Environmental Variables

- Environmental variables hold special values.
- Environmental variables are set by the system on initial login
  - **/etc/profile**, **~/.bash_profile** or **~/.profile**
- If you want to know what the variable holds call it with a "$" sign:
- **`env`** command

```
$ echo SHELL
SHELL
$ echo $SHELL
/bin/bash
$ echo $HOME
/cchome/arodrigu1
$ echo $PATH
/usr/X11R6/bin:/usr/local/bin:/bin:/
usr/bin
```

Ubuntu [Running]

jere@jere-VirtualBox: ~/test/script

```
SSH_AGENT_PID=1457
GPG_AGENT_INFO=/tmp/keyring-pPNXAg/gpg:0:1
TERM=xterm
SHELL=/bin/bash
XDG_SESSION_COOKIE=b9e0472ea390e1346b6eb81a00000008-1323772057.86708-1990901206
WINDOWID=58720261
GNOME_KEYRING_CONTROL=/tmp/keyring-pPNXAg
GTK_MODULES=canberra-gtk-module:canberra-gtk-module
USER=jere
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:su=3
34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;
1:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lz=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.
jar=01;31:*.rar=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.jpg=01;35:*.jpeg=01;35:
:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01
01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.m
nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:
.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.axv=01;35:*.anx=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36
6:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.axa=00;36:*.oga=00
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
SSH_AUTH_SOCK=/tmp/keyring-pPNXAg/ssh
SESSION_MANAGER=local/jere-VirtualBox:@/tmp/.ICE-unix/1395,unix/jere-VirtualBox:/tmp/.ICE-unix/1395
USERNAME=jere
DEFAULTS_PATH=/usr/share/gconf/ubuntu-2d.default.path
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu-2d:/etc/xdg
PATH=/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
DESKTOP_SESSION=ubuntu-2d
PWD=/home/jere/test/script
GNOME_KEYRING_PID=1386
LANG=en_US.UTF-8
MANDATORY_PATH=/usr/share/gconf/ubuntu-2d.mandatory.path
UBUNTU_MENUPROXY=libappmenu.so
```

Environmental Variables

- **$PATH**: The search path for commands
- Usually, we type in the commands in the following way:

```
$ ./hello.sh
Hello World
```

- By setting **`PATH=$PATH:~/scripts`** our working directory is included in the search path for commands, and we simply use the export command:

```
$ export PATH=$PATH:~/scripts
$ hello.sh
Hello World
```

Local Variables

- We can use variables as in any programming languages
- Stored as strings
- Declaring a variable:

```
$ STR='Hello World!'
$ echo $STR
Hello World!
```

- a value to a variable
- Call the variable by putting the '**$**' at the beginning

Double Quotes

- When assigning character data containing spaces or special characters, the data must be enclosed in either single or double quotes.
- Using double quotes (partial quoting) to show a string of characters will allow any variables in the quotes to be resolved.

```
$ var="test string"
$ new_var="Value of var is $var"
$ echo $new_var
Value of var is test string
```

Single Quotes

- Using single quotes (full quoting) to show a string of characters will not allow variable resolution.

```
$ newvar='Value of var is $var'
$ echo $newvar
Value of var is $var
```

Command Substitution

- The backquote "`" is different from the single quote "'".
  - It is used for command substitution: **`command`**
- You can assign the output of a command to a variable

```
$ ls
hello.sh myCleanDir.sh
$ LIST=`ls`
$ echo $LIST
hello.sh myCleanDir.sh
```

Conditional Statements

- Conditionals lets us decide whether to perform an action or not, this decision is taken by evaluating an expression

```
if [ expression ];        ## must have space between brackets
then
  statements
elif [ expression ];      ## brackets test an expression
then
  statements
else
  statements
fi
```

- The **`elif`** (else if) and else sections are optional.

Conditional Statements - Example

- Let's write a script that determines whether the word "UNIX" exists in the file "myfile"

grep: returns 0 if it finds something; returns non-zero otherwise

```
$ vi if1.sh
if grep "UNIX" myfile >/dev/null
then
  echo "It's there"
fi
$ ./if1.sh
It's there
```

redirect to /dev/null so that "intermediate" results do not get printed This file is available for everyone

Conditional Statements - Example

```
$ vi if2.sh
#!/bin/bash
if grep "UNIX" myfile >/dev/null
then
  echo "UNIX occurs in myfile"
else
  echo "No!"
  echo "UNIX does not occur in myfile"
fi
$ ./if2.sh
No! UNIX does not occur in myfile
```

Expressions

- Expressions can be:
  - String comparison
  - Numeric comparison
  - File operators
  - Logical operators

Expressions: String Comparisons

- String Comparisons:
  - **=** compare if two strings are equal
  - **!=** compare if two strings are not equal
  - **-n** evaluate if string length is greater than zero
  - **-z** evaluate if string length is equal to zero

Expressions: String Comparisons

- Examples:
  - **[s1 = s2]** (true if s1 same as s2, else false)
  - **[s1 != s2]** (true if s1 not same as s2, else false)
  - **[s1]** (true if s1 is not empty, else false)
  - **[-n s1]** (true if s1 has a length greater then 0, else false)
  - **[-z s2]** (true if s2 has a length of 0, otherwise false)

Expressions: String Comparisons

- Compare the user's name given with the environment variable $USER

```
$ vi if3.sh
#!/bin/bash
echo -n "Enter your login name: "  # ask user input
read name # store input in var
if [ "$name" = "$USER" ];
then
  echo "Hello, $name. How are you today ?"
else
  echo "You are not $USER, so who are you ?"
fi
$ ./if3.sh
Enter your login name: Jackie
You are not mftsai, so who are you ?
```

Expressions: Number Comparisons

- Number Comparisons:
  - **-eq** compare if two numbers are equal
  - **-ge** compare if one number is greater than or equal to a number
  - **-le** compare if one number is less than or equal to a number
  - **-ne** compare if two numbers are not equal
  - **-gt** compare if one number is greater than another number
  - **-lt** compare if one number is less than another number

Expressions: Number Comparisons

- Examples:
  - **[n1 -eq n2]** (true if n1 same as n2, else false)
  - **[n1-ge n2]** (true if n1 greater than or equal to n2, else false)
  - **[n1 -le n2]** (true if n1 less then or equal to n2, else false)
  - **[n1-ne n2]** (true if n1 is not same as n2, else false)
  - **[n1 -gt n2]** (true if n1 greater then n2, else false)
  - **[n1 -lt n2]** (true if n1 less then n2, else false)

Expressions: Number Comparisons

- Perform a mathematical operation if the number is between a range, otherwise let the user know the number entered is incorrect

```
$ vi if4.sh
#!/bin/bash
echo-n"Enter a number 1 < x < 10:"  #ask user
input
read num   # store input in var
if [ "$num" -lt 10 ]; then
  if [ "$num" -gt 1 ]; then
    echo "$num*$num=$(($num*$num))"
  else
    echo "Wrong insertion !"
  fi
else
  echo "Wrong insertion !"
fi
$ ./if4.sh
Enter a number 1 < x < 10: 5
5*5=25
```

Expressions: File Operators

- Files operators:
  - **-d** check if path given is a directory
  - **-f** check if path given is a file
  - **-s** check if path given is a symbolic link
  - **-e** check if file name exists
  - **-s** check if a file has a length greaterthan0
  - **-r** check if read permission is set for file or directory
  - **-w** check if write permission is set for a file or directory
  - **-x** check if execute permission is set for a file or directory

Expressions: File Operators

- Check if a certain file exists

```
$ vi if5.sh
#!/bin/bash
if [ -f /etc/passwd ]; then
  cp /etc/passwd .
  echo "Done."
else
  echo "This file does not exist."
  exit 1
fi
$ ./if5.sh
Done.
```

Expressions: Logical Operators

- Logical operators:
  - **&&** logically AND two logical expressions
  - || logically OR two logical expressions

for Loops

- Syntax:

```
for var in value1 value2 ...
do
  command_set
done
```

for Loops

- Lets calculate the smallest number among a set

```
$ vi for1.sh
#!/bin/bash
smallest=10000
for i in 5 8 19 8 7 3
do
  if [ $i -lt $smallest ]
  then
    smallest=$i
  fi done
echo $smallest
$ ./for1.sh
3
```

while Loops

- Syntax:

```
while [ expression ]
do
  command_set
done
```

while loop

- Lets do a summation of every number from 1 to 100

```
$ vi while1.sh
#! /bin/bash
i=1 # declare var
sum=0  # declare var
while [ $i -le 100 ]
do
  sum=`expr $sum + $i`
  i=`expr $i + 1`
done
echo The sum is $sum.
$ ./while1.sh
The sum is 5050.
```

Bash (Unix Shell)

- Example: for_example.sh

```bash
#!/usr/bin/env bash

for i in {1..5}
do
    echo "Welcome $i times"
done


echo "Bash version ${BASH_VERSION}..."
for i in {0..10..2}
do
    echo "Welcome $i times"
done
```

Bash (Unix Shell)

- Example: mkdirs.sh

```bash
#!/usr/bin/env bash

for i in {1..5}
do
    mkdir dir_$i
done
```

Bash (Unix Shell)

- Example: rename.sh

```bash
#!/usr/bin/env bash

for f in `ls ./data/unk_*`
do
    mv $f $f.txt
done
```

Bash (Unix Shell)

- Example: count.sh

```
#!/usr/bin/env bash

cat ./data/unk_list-* | cut -d ' ' -f1 | sort | uniq -c

```

Bash (Unix Shell)

- Example: average.sh

```bash
#!/usr/bin/env bash

cat ./data/baseline* | grep NDCG@10
cat ./data/baseline* | grep NDCG@10 | \
    awk '{ sum += $2 }; END { print "Average: " sum/NR }'
```

Customize Your Prompt

```
^_^ mftsai@ghost [~] ll
-bash: ll: command not found
O_O mftsai@ghost [~] ls
Codes      Documents  Dropbox        Library  Music    Pictures  Sites
Desktop    Downloads  Google Drive   Movies   Papers   Public    Tmp
^_^ mftsai@ghost [~]
```

Customize your Prompt to
Your_ID@host
Try to make it colorful!

Ex: 100001@ghost

- Hint: Revise ~/.bashrc

PS1="\u@ghost [\w] "

or

PS1="\`if [ \$? = 0 ]; then echo \[\e[33m\]^_^\[\e[0m\]; else echo \[\e[31m\]O_O\[\e[0m\]; fi\` \u@ghost [\w] "

Bash (Unix Shell)

- Useful Tutorials
  - Bash Programming
  - Bash by example
  - Bash Scripting Tutorial

Bash (Unix Shell)

- Further Information
  - grep tutorial
  - sed tutorial
  - awk tutorial

Python

Python

- An interpreted, general-purpose high-level programming language
- Design philosophy is to emphasize code readability
    - Use of indentation for block delimiters
- Used as a scripting language, but is also used in a wide range of non-scripting contexts

Python

- Download
  - http://www.python.org/getit/
- Useful Tutorials
  - Victor’s Python 教學
  - The Python Tutorial

Python

- Example: list_comprehension.py

```
# Define a function to construct list

S = [x**2 for x in range(10)]
V = [2**i for i in range(13)]
M = [x for x in S if x % 2 == 0]

print S; print V; print M
```

Python

- Example: list_comprehension.py

```python
# Define a function to construct list

S = [x**2 for x in range(10)]
V = [2**i for i in range(13)]
M = [x for x in S if x % 2 == 0]

print S; print V; print M
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
[0, 4, 16, 36, 64]
```

Python

- Example: list_comprehension.py

```python
# List comprehensions provide a
# concise way to create lists

vec = [2, 4, 6]
print [3*x for x in vec]
print [3*x for x in vec if x > 3]
print [3*x for x in vec if x < 2]
print [[x,x**2] for x in vec]
print [(x, x**2) for x in vec]

vec1 = [2, 4, 6]
vec2 = [4, 3, -9]
print [x*y for x in vec1 for y in vec2]
print [x+y for x in vec1 for y in vec2]
print [vec1[i]*vec2[i] for i in range(len(vec1))]
print [str(round(355/113.0, i)) for i in range(1,6)]
```

Python

- Example: list_comprehension.py

```python
# List comprehensions provide a
# concise way to create lists

vec = [2, 4, 6]
print [3*x for x in vec]
print [3*x for x in vec if x > 3]
print [3*x for x in vec if x < 2]
print [[x,x**2] for x in vec]
print [(x, x**2) for x in vec]

vec1 = [2, 4, 6]
vec2 = [4, 3, -9]
print [x*y for x in vec1 for y in vec2]
print [x+y for x in vec1 for y in vec2]
print [vec1[i]*vec2[i] for i in range(len(vec1))]
print [str(round(355/113.0, i)) for i in range(1,6)]
```

```
[6, 12, 18]
[12, 18]
[]
[[2, 4], [4, 16], [6, 36]]
[(2, 4), (4, 16), (6, 36)]
[8, 6, -18, 16, 12, -36, 24, 18, -54]
[6, 5, -7, 8, 7, -5, 10, 9, -3]
[8, 12, -54]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

Python

- Example: list_comprehension.py

```
#First build a list of non-prime numbers, using a single list comprehension,
#then use another list comprehension to get the "inverse" of the list,
#which are prime numbers.

noprimes = [j for i in range(2, 8) for j in range(i*2, 50, i)]
primes = [x for x in range(2, 50) if x not in noprimes]
print primes
```

Python

- Example: list_comprehension.py

```python
#First build a list of non-prime numbers, using a single list comprehension,
#then use another list comprehension to get the "inverse" of the list,
#which are prime numbers.

noprimes = [j for i in range(2, 8) for j in range(i*2, 50, i)]
primes = [x for x in range(2, 50) if x not in noprimes]
print primes
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Python

- Useful Packages
  - Math: numpy
  - Plot: matplotlib
  - Scientific Computing: scipy
  - ==> pylab (similar to matlab)
- Demos
  - pylab_finance.py
  - pylab_surface3d.py

Perl

Perl

- A high-level, general-purpose, interpreted, dynamic programming language
- Borrows features from other programming languages including C, shell scripting (sh), awk, and sed
- Provides powerful text processing facilities
  - Used for a wide range of tasks including system administration, web development, network programming, games, bioinformatics, and GUI development

Perl

- Download
  - http://www.perl.org/get.html
- Useful tutorials
  - Perl 學習手札
  - Perl Tutorial

Perl

- Example: frequency.pl

```perl
#!/usr/bin/perl

use warnings;
use strict;

sub frequency {
    my $text = join('', @_);
    my %letters;
    foreach (split //, $text) {
        $letters{$_}++;
    }
    return %letters;
}

my $text = "This is the start of a tutorial on Perl!!";

my %count = frequency($text);

foreach (sort keys %count) {
    print "\t", $count{$_}, " '$_'", ($count{$_} == 1)? "": "s", "\n";
}
```

Perl

- Example: simple_data_structure.pl

Perl

- Example: simple_data_structure.pl

```
# A Stack

print "Making a Stack\n";
@stack = qw( awk bash chmod );
print "Initial stack:\n  @stack \n";
push (@stack, "diff");
print "Push item on stack:\n  @stack \n";
$item = "Emacs";
push (@stack, $item);
print "Push item on stack:\n  @stack \n";
$top = pop @stack;
print "Popping top of stack:  $top\n";
print "Final stack:\n  @stack \n\n";
```

```
Making a Stack
Initial stack:
  awk bash chmod
Push item on stack:
  awk bash chmod diff
Push item on stack:
  awk bash chmod diff Emacs
Popping top of stack: Emacs
Final stack:
  awk bash chmod diff
```

Perl

- Example: simple_data_structure.pl

```
# A Stack

print "Making a Stack\n";
@stack = qw( awk bash chmod );
print "Initial stack:\n  @stack \n";
push (@stack, "diff");
print "Push item on stack:\n  @stack \n";
$item = "Emacs";
push (@stack, $item);
print "Push item on stack:\n  @stack \n";
$top = pop @stack;
print "Popping top of stack:  $top\n";
print "Final stack:\n  @stack \n\n";
```

```
Making a Stack
Initial stack:
  awk bash chmod
Push item on stack:
  awk bash chmod diff
Push item on stack:
  awk bash chmod diff Emacs
Popping top of stack: Emacs
Final stack:
  awk bash chmod diff
```

```
# A Queue

print "Making a \"First In First Out\" Queue\n";
@queue = qw( lpr mcopy ps );
print "Initial queue:\n  @queue \n";
unshift(@queue, "kill");
print "Add item to queue:\n  @queue \n";
$item = "df";
unshift(@queue, $item);
print "Add item to queue:\n  @queue \n";
$fifo = pop @queue;
print "Remove FIFO item: $fifo\n";
print "Final queue:\n  @queue \n\n";
```

```
Making a "First In First Out" Queue
Initial queue:
  lpr mcopy ps
Add item to queue:
  kill lpr mcopy ps
Add item to queue:
  df kill lpr mcopy ps
Remove FIFO item: ps
Final queue:
  df kill lpr mcopy
```

Perl

- Example: simple_data_structure.pl

```
# Linked Lists

print "Making Linked Lists\n";
## Method #1 using 2D Arrays

sub print_list {
  $max = $_[0];
  for ($i=0; $i<$max; $i++)
  {
    print "$i.  $list[$i][0]\t $list[$i][1]\n";
  }
}

# Declaring a 2-D Array, which is just an array of 1-D arrays

@list = ( ["vi   ", "Null"], ["emacs", "Null"], ["joe  ", "Null" ]);

$max = $#list + 1;

print "Initial Values\n";
print_list($max);

print "\n\n";
```

```
Making Linked Lists
Initial Values
0. vi         Null
1. emacs      Null
2. joe        Null
```

Perl

- Example: simple_data_structure.pl

```
## Method #3 - Using a Hash

print "Using a Hash\n";

# Initializing a hash using the "correspond" operator to make easy reading

%hash = (
          "man" =>  "Get UNIX Help:more",
          "cat" => "Display Files:Null",
          "more"=> "Page Through Files:cat");

print "Traversing list:\n";
$next = "man";
while ($next !~ "Null")
  { @data = split(/:/, $hash{$next});
    print "$next  $data[0] \n";
    $next = $data[1];
  }

 print "\n\n";
```

```
Using a Hash
Traversing list:
man     Get UNIX Help
more    Page Through Files
cat     Display Files
```

Perl

- Regular Expression

Perl

- Regular Expression

Code	Meaning
\w	Alphanumeric Characters
\W	Non-Alphanumeric Characters
\s	White Space
\S	Non-White Space
\d	Digits
\D	Non-Digits
\b	Word Boundary
\B	Non-Word Boundary
\A or ^	At the Beginning of a String
\Z or $	At the End of a String
.	Match Any Single Character

Perl

- Regular Expression

Code	Meaning
\w	Alphanumeric Characters
\W	Non-Alphanumeric Characters
\s	White Space
\S	Non-White Space
\d	Digits
\D	Non-Digits
\b	Word Boundary
\B	Non-Word Boundary
\A or ^	At the Beginning of a String
\Z or $	At the End of a String
.	Match Any Single Character

Code	Meaning
*	Zero or More Occurrences
?	Zero or One Occurrence
+	One or More Occurrences
{ N }	Exactly N Occurrences
{ N,M }	Between N and M Occurrences
.* <thingy>	Greedy Match, up to the **last** *thingy*
.*? <thingy>	Non-Greedy Match, up to the **first** *thingy*
[set_of_things]	Match Any Item in the Set
[^ set_of_things]	Does Not Match Anything in the Set
(some_expression)	Tag an Expression
$1..$N	Tagged Expressions used in Substitutions

Perl

- Example: regular_expression.pl

Perl

- Example: regular_expression.pl

```
sub grep_pattern        # Print strings which contain the pattern
{ foreach (@strings)
    {print "$_\n" if /$pattern/;
     }
print "\n\n";
}
```

Perl

- Example: regular_expression.pl

```
sub grep_pattern        # Print strings which contain the pattern
{ foreach (@strings)
    {print "$_\n" if /$pattern/;
     }
print "\n\n";
}
```

```
### Setting up the Array of strings

@strings = ("Two, 4, 6, Eight", "Perl is cryptic", "Perl is great");

@strings[3..6] = ("1, Three", "Five, 7", "Write in Perl", "Programmer's heaven");
 print_array;
```

Two, 4, 6, Eight
Perl is cryptic
Perl is great
1, Three
Five, 7
Write in Perl
Programmer's heaven

Perl

- Example: regular_expression.pl

Perl

- Example: regular_expression.pl

```
## Find the word "Perl"
$pattern = 'Perl';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **Perl**
Perl is cryptic
Perl is great
Write in **Perl**

Perl

- Example: regular_expression.pl

```
## Find the word "Perl"
$pattern = 'Perl';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **Perl**
Perl is cryptic
Perl is great
Write in **Perl**

```
## Find "Perl" at the beginning of a line
$pattern = '^Perl';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **^Perl**
Perl is cryptic
Perl is great

Perl

- Example: regular_expression.pl

```
## Find the word "Perl"
$pattern = 'Perl';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **Perl**
Perl is cryptic
Perl is great
Write in **Perl**

```
## Find sentences that contain an "i"
$pattern = 'i';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **i**
Two, 4, 6, E**i**ght
Perl **i**s cryptic
Perl **i**s great
F**i**ve, 7
Wr**i**te in Perl

```
## Find "Perl" at the beginning of a line
$pattern = '^Perl';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **^Perl**
Perl is cryptic
Perl is great

Perl

- Example: regular_expression.pl

```
## Find the word "Perl"
$pattern = 'Perl';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **Perl**
Perl is cryptic
Perl is great
Write in **Perl**

```
## Find sentences that contain an "i"
$pattern = 'i';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **i**
Two, 4, 6, **Eight**
Perl **is** cryptic
Perl **is** great
Five, 7
Write in Perl

```
## Find "Perl" at the beginning of a line
$pattern = '^Perl';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **^Perl**
Perl is cryptic
Perl is great

```
## Find words starting in "i", i.e. a space preceeds the letter
$pattern = '\si';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **\s i**
Perl **is** cryptic
Perl **is** great
Write **in** Perl

Perl

- Example: regular_expression.pl

Perl

- Example: regular_expression.pl

```
## Find strings containing a digit
$pattern = '\d';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \d
Two, **4**, 6, Eight
1, Three
Five, **7**

Perl

- Example: regular_expression.pl

```
## Find strings containing a digit
$pattern = '\d';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \d
Two, 4, 6, Eight
1, Three
Five, 7

```
## Find strings with a digit at the end of a line
$pattern = '\d+$';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \d+ $
Five, 7

Perl

- Example: regular_expression.pl

```
## Find strings containing a digit
$pattern = '\d';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \d
Two, **4**, **6**, Eight
1, Three
Five, **7**

```
## Search for a digit, possible stuff in between, and another digit
$pattern = '\d.*\d';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \d .* \d
Two, **4, 6**, Eight

```
## Find strings with a digit at the end of a line
$pattern = '\d+$';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \d+ $
Five, **7**

Perl

- Example: regular_expression.pl

```
## Find strings containing a digit
$pattern = '\d';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \d
Two, **4**, **6**, Eight
1, Three
Five, **7**

```
## Search for a digit, possible stuff in between, and another digit
$pattern = '\d.*\d';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \d .* \d
Two, **4, 6**, Eight

```
## Find strings with a digit at the end of a line
$pattern = '\d+$';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \d+ $
Five, **7**

```
## Search for a digit followed by some stuff
$pattern = '\d+.+';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \d+ .+
Two, **4, 6**, Eight
1, Three

Perl

- Example: regular_expression.pl

Perl

- Example: regular_expression.pl

```
## Find four-letter words, i.e. four characters offset by word boundaries
$pattern = '\b\w{4}\b';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: \b \w{4} \b
Perl is cryptic
Perl is great
Five, 7
Write in **Perl**

Perl

- Example: regular_expression.pl

```
## Find four-letter words, i.e. four characters offset by word boundaries
$pattern = '\b\w{4}\b';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **\b \w{4} \b**
Perl is cryptic
Perl is great
Five, 7
Write in **Perl**

```
## Sentences with three words, three word fields separated by white space
$pattern = '\w+\s+\w+\s+\w+';
print "Searching for: $pattern\n";
grep_pattern;
```

Searching for: **\w+ \s+ \w+ \s+ \w+**
Perl is cryptic
Perl is great
Write in Perl

Perl

- Example: regular_expression.pl

Perl

- Example: regular_expression.pl

```
## Sentences with three words, add "n't" after the middle word
$pattern = '(\w+\s+)(\w+)(\s+\w+)';
print "Searching for: $pattern\n";
foreach(@strings)
  {
       s/$pattern/$1$2n\'t$3/;
  }
print_array;
```

Searching for: (\w+ \s+) (\w+) (\s+ \w+)
Two, 666, 444, Amazing
Pascal **isn't** cryptic
Pascal **isn't** Amazing
111, Amazing
Five, 777
Amazing **isn't** Cobol
Programmer heaves

Perl

- Example: regular_expression.pl

```
## Sentences with three words, add "n't" after the middle word
$pattern = '(\w+\s+)(\w+)(\s+\w+)';
print "Searching for: $pattern\n";
foreach(@strings)
  {
       s/$pattern/$1$2n\'t$3/;
  }
print_array;
```

Searching for: (\w+ \s+) (\w+) (\s+ \w+)
Two, 666, 444, Amazing
Pascal **isn't** cryptic
Pascal **isn't** Amazing
111, Amazing
Five, 777
Amazing **isn't** Cobol
Programmer heaves

```
## Sentences with either an "o" or an "e" in them
$pattern = '[oe]';
print "Searching for: $pattern\n";
foreach(@strings)
  {
       s/$pattern/x/g;   # The "g" modifyer means "global", or replace all
  }                      # occurrences of the "o" or "e" found on that line.
print_array;
```

Searching for: **[oe]**
Twx, 666, 444, Amazing
Pascal isn't cryptic
Pascal isn't Amazing
111, Amazing
Fivx, 777
Amazing isn't **Cxbxl**
Prxgrammxr hxavxs

Git

Git

- A distributed revision control system
- Initially designed and developed by Linus Torvalds for Linux kernel development
- Git is free software distributed under the terms of the GNU General Public License version 2
- History of revision control system
    - CVS --> SVN --> Git

Git

- Download
  - http://git-scm.com/download
- Useful Tutorials
  - 寫給大家的 Git 教學
  - Git Reference
  - A Visual Git Reference

GitHub

- A web-based hosting service for software development projects that use the Git revision control system
- Provides social networking functionality such as feeds, followers and the network graph to display how developers work on their versions of a repository
- https://github.com/

CP2 Final Exams

- Location and Time: 06/14 (Wed) at 9:10am in PC Lab
- Please arrive 15 minutes in advance!!
- Durations
  - Writing Exam: 1 hour (from 9:10 to 10:10)
  - Coding Exam: 2 hours (from 10:15 to 12:30)
- Writing Exam
  - NO open book!!
  - 6 questions, 100 points
- Coding Exam
  - Allow one A4 hand-written page, such as code snippets
  - 6 questions, 120 points

Course Review

- Basic Data Structure
  - Linked List
  - Stack
  - Queue
  - Tree
    - Tree traverse
- Basic Algorithms
  - Sorting
    - Quick sort, Merge sort, Insertion sort

One More Thing...

- 『唯有終生學習，才能繼續突破，不斷地解決問題。』
  - 政大資科系友專訪--李致緯
  - 台灣資訊領域學生盛會 SITCON：從突破開始
- MIT - Introduction to Algorithms
- Coursera
  - Stanford - Algorithms: Design and Analysis
  - University of Washington: Programming Languages