# FUNCTIONAL PROGRAMMING
# NO.2 FUNCTION AND LIST

Tatsuya Hagino
hagino@sfc.keio.ac.jp

lecture slide URL

https://vu5.sfc.keio.ac.jp/slide/

## Program Development Environment

- CUI vs GUI Program Development Environment
  - CUI (Character User Interface) or CLI (Command Line Interface)
    - simple and light weight
    - compiler and library only
    - use text editor to write codes
  - GUI (Graphical User Interface)
    - modern but heavy
    - editor, compiler, debugger, and other tools are integrated
    - ex. eclipse, Xcode, Visual Studio

- CUI
  - UNIX (Linux): shell (sh, csh, tcsh, bash)
  - Mac OS X: Terminal
  - Windows: command prompt

- Text Editor
  - UNIX (Linux): vi (vim), emacs
  - Mac OS X: TextEdit, mi, emacs
  - Windows: notepad, xyzzy

# UNIX Basic Commands

- CUI Basic
  - type command to execute
  - give command name and arguments
  - correctly set the current working directory
  - folder = directory

$$\%\ \text{command}\ \underbrace{\text{arg1 arg2 arg3}}_{\text{arguments}}$$

prompt

- Basic commands for shell (UNIX or Mac OS X)
  - Mac OS X is based on UNIX.
  - cygwin may be installed for Windows.

| command | meaning |
|---|---|
| pwd | Print current working directory |
| cd *dir* | Change directory to *dir* |
| ls *dir* | List files in *dir* |
| ls -l *dir* | List files in *dir* with information |
| cat *file* | Show the content of *file* |
| more *file* | Show the content of *file* page by page |
| mkdir *dir* | Create a new directory *dir* |
| rmdir *dir* | Delete directory *dir* |
| rm *file* | Delete (remove) *file* |
| *command* < *file* | *command* takes input from *file* |
| *command* > *file* | *command* outputs the result to *file* |

no undelete
deleted file cannot be recovered

## Show the content of a file

- Write a Haskell program similar to unix cat command
  - It outputs the content of a given file.

cat.hs

```
main = do cs <- getContents
          putStr cs
```

```
% stack ghc cat.hs
...
% ./cat < cat.hs
main = do cs <- getContents
          putStr cs
%
```

- "./" is the current directory
- "./cat" is the "cat" program in the current directory
- For windows, replace "/" with "¥"

# cat Program

- getContents
    - It is an action.
    - When this action is evaluated, the standard input is read.
    - It returns a string of the whole standard input.

- putStr cs
    - Returns an action which outputs the string cs to the standard output.

- do expression
    - Multiple actions are evaluated one line by one line from top to bottom
    - If one the action fails, stops
    - cs <- getContents
        - The evaluation result of action getContents is bound to variable cs
        - cs can be used in the following lines

```
main = do cs <- getContents
          putStr cs
```

Layout syntax

- lines with same indent are grouped
- block structure
- { and } are not necessary

## Lazy Evaluation

cat.hs

```
main = do cs <- getContents
          putStr cs
```

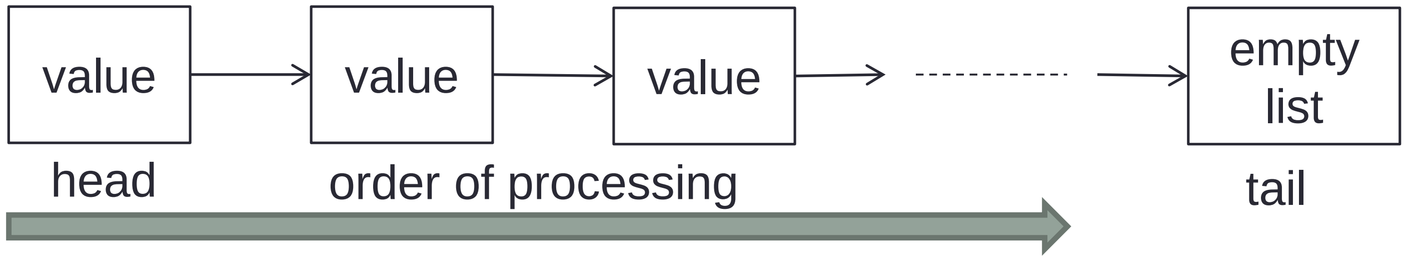
- getContents does not read the standard input at once
  - cs is a string which represents the standard input
  - The actual content of cs is read from the standard input when it is referred.
- putStr accesses the content of cs, and triggers the real read from the standard input
  - The amount putStr wants is read.
  - From the terminal, only one line is read at once.

lazy evaluation ⇔ eager evaluation

Evaluate later when it is required
Try not to evaluate

Evaluate first
Evaluate everything

# List



- Connecting multiple values
- Process from the head to the tail
- The last value is 'empty list'.
  - Similar to NULL pointer in C

- A list can hold the same kind of values.
  - No mixing different type values (i.e. integer and character)
- Cannot change the value
  - Not like array in C

## List Literal

- [1, 2, 3]
  - List of number 1, 2 and 3
- ["aa", "bb", "cc"]
  - List of three strings
- ['a', 'b', 'c']
  - List of three characters
  - Same as "abc"
- Only one kind (i.e. type) of data in a same list
  - [1, 'c', "string"] is wrong
  - [1, [2, [3]]] is wrong
- []
  - empty list

## Count the number of lines in a file

countline.hs

```haskell
main = do cs <- getContents
          print $ length $ lines cs
```

- Try the above program.

```
% stack ghc countline.hs
...
% ./countline < countline.hs
2
%
```

## countline details

countline.hs

```
main = do cs <- getContents
          print $ length $ lines cs
```

- '`$`' operator
  - Binary operator like '`+`' and '`*`'
  - '`x $ y`' means '`x(y)`'
  - '`length $ lines cs`' is '`length(lines cs)`'
  - '`print $ length $ lines cs`' is
    - `print(length(lines cs))`
  - '`print length lines cs`' is
    - `(((print length) lines) cs)`

# countline details (continue)

- '`lines`' function
  - Divide the string by lines
  - `lines "aaa\nbbb\nccc\n"` → `["aaa", "bbb", "ccc"]`
  - `lines "aaa\n"` → `["aaa"]`
  - `lines "aaa"` → `["aaa"]`
  - `lines "\n"` → `[""]`
  - `lines ""` → `[]`

- '`length`' function
  - Returns the number of elements of the list
  - `length [1, 2, 3, 4] → 4`
  - `length [5, 11] → 2`
  - `length []` → `0`
  - `length ["aa", "bb"] →` `2`
  - `length ["aa"] →` `1`
  - `length [""] →` `1`
  - `length "string"` → `6`
  - `length "str"` → `3`
  - `length ""` → `0`

- '`print`' function
  - Returns an action to output the value
  - The value is serialized to a string.

# USA-states.txt

- USA state names, their abbreviation and their capitals
  - See http://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_United_States

tab

```
USA-states.txt

AK      Alaska  Juneau
AL      Alabama         Montgomery
AR      Arkansas        Little Rock
AZ      Arizona         Phoenix
CA      California      Sacramento
CO      Colorado        Denver
...
WV      West VirginiaCharleston
WY      Wyoming         Cheyenne
```

- Items are separated by tabs.
- Available from

  **https://web.sfc.keio.ac.jp/~hagino/fp20/USA-states.txt**

## Show the first 10 lines of the file

head.hs

```
main = do cs <- getContents
          putStr $ firstNLines 10 cs

firstNLines n cs = unlines $ take n $ lines cs
```

- Try the above program.

```
% stack ghc head.hs
...
% ./head < USA-states.txt
AK      Alaska  Juneau
AL      Alabama Montgomery
AR      Arkansas        Little Rock
AZ      Arizona Phoenix
CA      California      Sacramento
CO      Colorado        Denver
CT      Connecticut     Hartford
DE      Delaware        Dover
FL      Florida Tallahassee
GA      Georgia Atlanta
```

## Application of arguments to a function

***func arg1 arg2 ・・・・***

- Applying an argument to a function
  - *func arg*

- With two arguments
  - *func arg1 arg2*

- With three arguments
  - *func arg1 arg2 arg3*

- Parenthesizes are not necessary
  - *func arg1 arg2* → ((*func arg1*) *arg2*)
  - *func arg1 arg2 args* → (((*func arg1*) *arg2*) *arg3*)

*func arg1 arg2* $\neq$ *func*（*arg1*, *arg2*）

## Defining a function

***func param1 param2 ･･･ = body***

- **`firstNLines n cs = unlines $ take n $ lines cs`**
  - Defining '**`firstNLines`**'
  - '**`firstNLines`**' takes two parameters '**`n`**' and '**`cs`**'
  - The parameters can be referred in the body
  - Its body is '**`unlines $ take n $ lines cs`**'

## '`unlines`' and '`take`'

- '**`unlines`**' function
  - Reverse of '**`lines`**' function.
  - Concatinate strings in a list.
  - unlines ["aaa", "bbb", "ccc"] → **`"aaa\nbbb\nccc\n"`**
  - **`unlines ["aaa"]`** → **`"aaa\n"`**
  - **`unlines [""]`** → **`"\n"`**
  - **`unlines []`** → **`""`**
  - **`unlines ["aaa\n"]`** → **`["aaa\n\n"]`**

- '**`take n`**' function
  - Returns a list consists of first **n** elements from the list.
  - If the length of the list is less than **n**, returns the list itself.
  - **`take 3 [5, 2, 4, 6, 8]`** → **`[5, 2, 4]`**
  - **`take 3 [5]`** → **`[5]`**
  - **`take 3 []`** → **`[]`**
  - **`take 3 "string"`** → **`"str"`**
  - **`take 0 [1, 2, 3]`** → **`[]`**

## Exercise 2-1

head.hs

```
main = do cs <- getContents
          putStr $ firstNLines 10 cs

firstNLines n cs = unlines $ take n $ lines cs
```

- Rewrite the above program without using '**$**'.

Note

You can only use functions or techniques taught in this course.
Please do not copy programs on the internet.

# 'reverse' and 'words'

- 'reverse' function
  - Reverse the list.
  - `reverse [1, 2, 3]` → `[3, 2, 1]`
  - `reverse []` → `[]`
  - `reverse "string"` → `"gnirts"`
  - `reverse ""` → `""`
  - `reverse ["abc", "def", "ghi"]`
    → `["ghi", "def", "abc"]`

- 'words' function
  - Divide the string into a list of words.
  - Blanks (including tabs and carriage returns) are separators.
  - `words "This is a pen."` → `["This", "is", "a", "pen."]`
  - `words " a(1, 2, 3) "` → `["a(1,", "2,", "3)"]`
  - `words "a\nb\nc\n"` → `["a", "b", "c"]`
  - `words ""` → `[]`

## Exercise 2-2

reverse.hs

```
main = do cs <- getContents
          putStr $ reverseLines cs

reverseLines cs = ...
```

- Complete the above program which reverse the lines in a file.

```
% stack ghc reverse.hs
...
% ./reverse < USA-states.txt
WY      Wyoming  Cheyenne
WV      West Virginia    Charleston
WI      Wisconsin        Madison
WA      Washington       Olympia
VT      Vermont  Montpelier
...
AK      Alaska   Juneau
```

## Exercise 2-3

tail.hs

```
main = do cs <- getContents
          putStr $ lastNLines 10 cs

lastNLines n cs = unlines $ takeLast n $ lines cs

takeLast n ss = ...
```

- Complete the above program which outputs the last 10 lines of the file.

```
% stack ghc tail.hs
...
% ./tail < USA-states.txt
SD      South Dakota    Pierre
TN      Tennessee       Nashville
TX      Texas   Austin
UT      Utah    Salt Lake City
VA      Virginia Richmond
VT      Vermont Montpelier
WA      Washington      Olympia
WI      Wisconsin       Madison
WV      West Virginia   Charleston
WY      Wyoming Cheyenne
```

## Exercise 2-4 and 2-5

countbyte.hs

```
main = do cs <- getContents
          print ...
```

- Count the number of bytes in a file.

countword.hs

```
main = do cs <- getContents
          print ...
```

- Count the number of words in a file.

## Exercise 2-6

abbr.hs

```
main = do cs <- getContents
          putStr $ ...
```

- Show the first 5 lines followed by “...” and the last 5 lines.
- Use ‘++’ to concatenate two lists together.

```
% stack ghc abbr.hs
...
% ./abbr < USA-states.txt
AK         Alaska    Juneau
AL         Alabama   Montgomery
AR         Arkansas  Little Rock
AZ         Arizona   Phoenix
CA         California          Sacramento
...
VA         Virginia  Richmond
VT         Vermont   Montpelier
WA         Washington          Olympia
WI         Wisconsin Madison
WV         West Virginia       Charleston
WY         Wyoming   Cheyenne
```

## Summary of functions and actions

| function | example | description |
|---|---|---|
| **`putStr`** | **`putStr cs`** | Returns an action of outputting sting **`cs`** |
| **`putStrLn`** | **`putStrLn cs`** | Returns an action of outputting string **`cs`** and a carriage return |
| **`print`** | **`print x`** | Returns an action of outputting value **`x`** |
| **`length`** | **`length xs`** | Returns the length of list **`xs`** |
| **`take`** | **`take n xs`** | Returns the first **`n`** elements from list **`xs`** |
| **`reverse`** | **`reverse xs`** | Returns the reverse of list **`xs`** |
| **`lines`** | **`lines cs`** | Divides string **`cs`** into the list of lines |
| **`unlines`** | **`unlines xs`** | Concatenates strings in list **`xs`** by adding carriage returns |
| **`words`** | **`words cs`** | Divides string **`cs`** into the list of words |

| action | example | description |
|---|---|---|
| **`getContents`** | **`getContents`** | An action of reading the standard input |