Introduction to Computers and Programming LAB-10 2016/12/07

✧ The output must be in our sample output format.

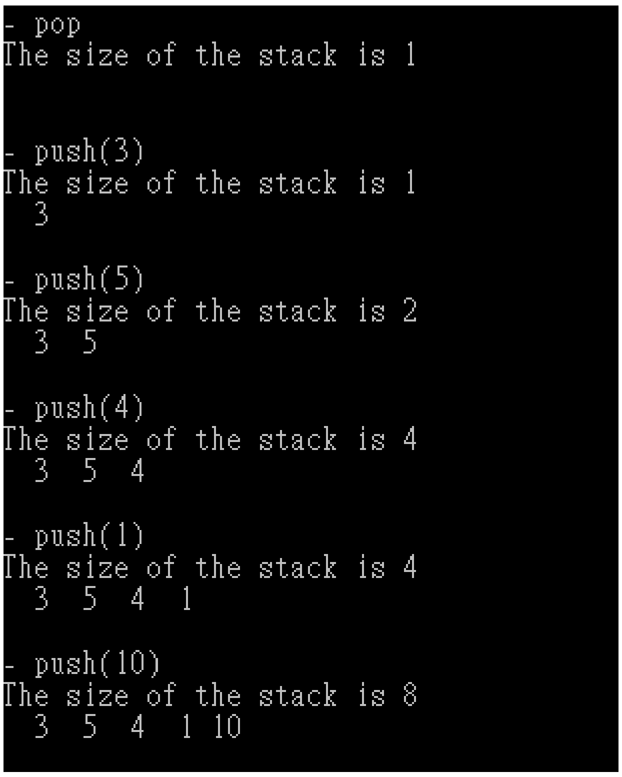
✧ You can raise your hand to demo once you finish a program.

✧ The bonus question is for this lab only. If you cannot finish question 1 to 3 in time, you are allowed to demo them at next lab hours.

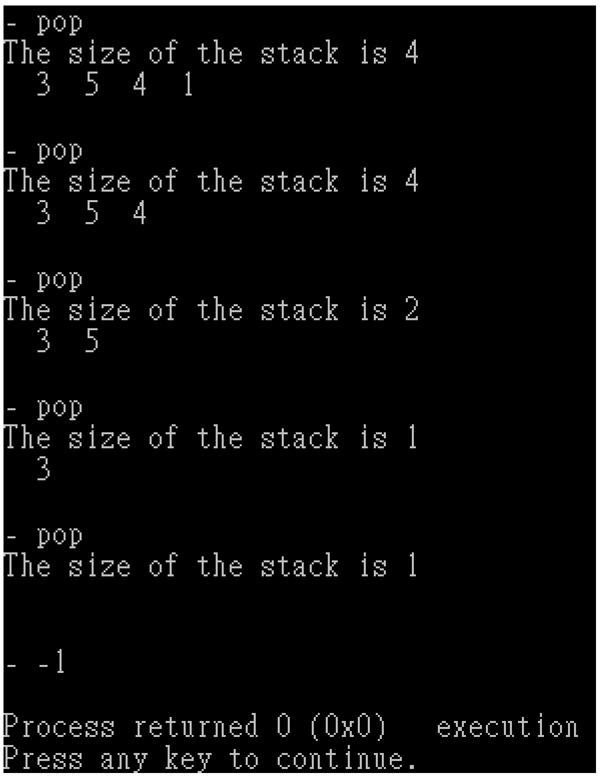
✧ TAs will update lab records every Monday after the lab hours in the link: http://goo.gl/ZVJu2Y

1. Dynamic Stack

Implement a dynamic non-negative integer stack that has the initial size 1, and will append or shrink if necessary. There are **two operations: *push(n)*, *pop***. If the space isn't enough to push an element, append the stack with double size. If there is a half of stack is empty after pop an element, shrink the stack to half size. Be aware that the minimal size of the stack is 1. After performing a push(n) or pop operation, show the information (the size and contents) of the stack.
If the input is -1, exit the program. Please use *malloc/calloc/realloc/free* functions to implement the dynamic stack.



```
- pop
The size of the stack is 1


- push(3)
The size of the stack is 1
  3

- push(5)
The size of the stack is 2
  3  5

- push(4)
The size of the stack is 4
  3  5  4

- push(1)
The size of the stack is 4
  3  5  4  1

- push(10)
The size of the stack is 8
  3  5  4  1 10
```



```
- pop
The size of the stack is 4
  3  5  4  1

- pop
The size of the stack is 4
  3  5  4

- pop
The size of the stack is 2
  3  5

- pop
The size of the stack is 1
  3

- pop
The size of the stack is 1



- -1

Process returned 0 (0x0)   execution
Press any key to continue.
```

2. Linked List

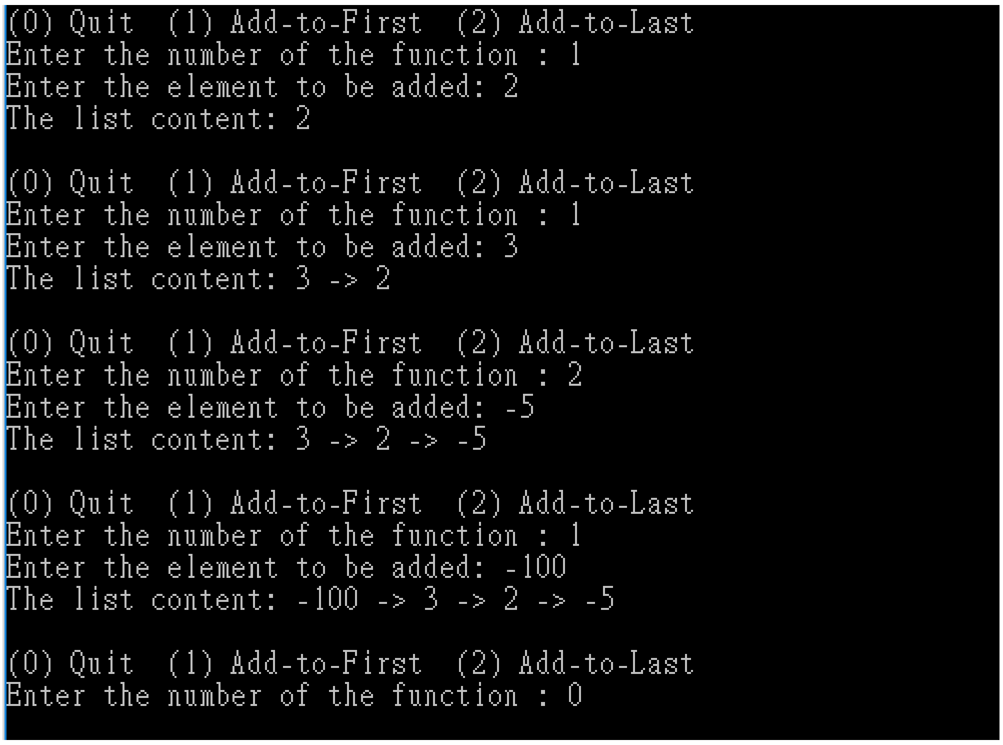
Write a program to implement integer linked lists. You have to use the linked list to store the user's input. The program will ask users to enter the number of function, and the element are going to be added. It has functions as listed below:

(0) ***Exit***: close the program.

(1) ***Add-to-First***: add the element as the first element in the list.

(2) ***Add-to-Last***: add the element as the last element in the list.

After performing those two add functions, the program will show the contents of the list.



```
(0) Quit  (1) Add-to-First  (2) Add-to-Last
Enter the number of the function : 1
Enter the element to be added: 2
The list content: 2

(0) Quit  (1) Add-to-First  (2) Add-to-Last
Enter the number of the function : 1
Enter the element to be added: 3
The list content: 3 -> 2

(0) Quit  (1) Add-to-First  (2) Add-to-Last
Enter the number of the function : 2
Enter the element to be added: -5
The list content: 3 -> 2 -> -5

(0) Quit  (1) Add-to-First  (2) Add-to-Last
Enter the number of the function : 1
Enter the element to be added: -100
The list content: -100 -> 3 -> 2 -> -5

(0) Quit  (1) Add-to-First  (2) Add-to-Last
Enter the number of the function : 0
```

3. Profile Table

Design a program that can make a table which contains the profiles of the workers in the company, and can do the four function below:

Functions:

(1) **assign**: When types **assign**, the program needs to go into an assign mode that can repeatedly add new profile, until input an empty line.

(2) **show**: When types **show**, the program needs to show the table of the profiles.

(3) **sort** *order_kind*: When types **sort**, the program needs to show the table of the profiles in the order of *order_kind.* There are 4 *order_kind*:

- (a) height: sort the profile from the tallest to the smallest.
- (b) weight: sort the profile from the lightest to the heaviest.
- (c) title: sort the profile from the highest to the lowest. (ceo>manager>employee)
- ● **sort** function is just showing the table of profile in the order that it asked, it doesn't change the position of the original profile table. It means that the table must be **show** in the original order no matter how much time the **sort** works.

(4) **clear**: When types **clear**, the program needs to clear the table of the profiles.

assign data format:

<Name>	<Height>	<Weight>	<E-mail>
Alice	165	51	Alice11@manager.com

output table format:

Name	height	weight	id	title
Alice	165	51	Alice11	manager

Notes:

(1) You need to deal the space between the inputs.

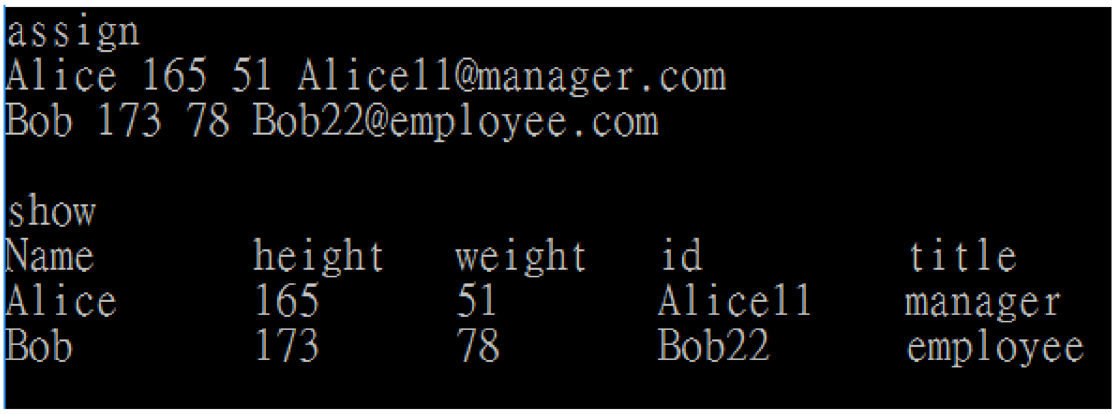
(2) id is the part of E-mail before '@'.

(3) title is the part of E-mail between '@' and the first '.'

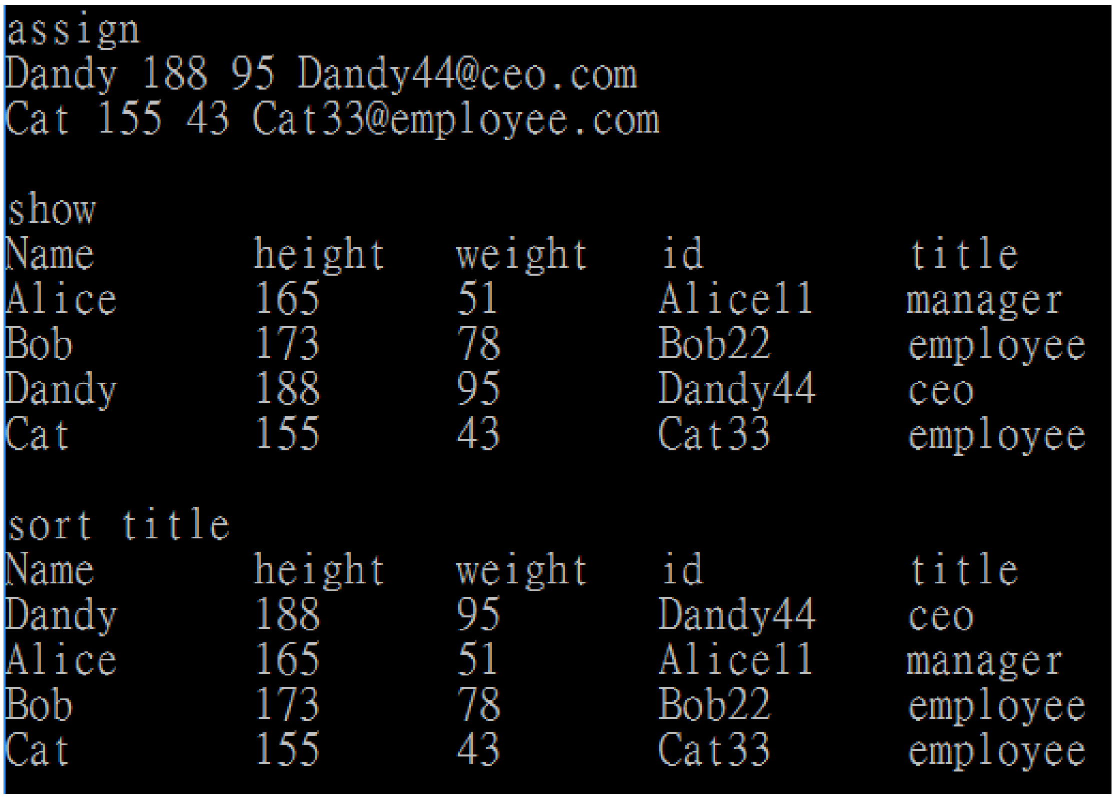
(4) The **Name** in the profile table cannot be repeated.

Examples:

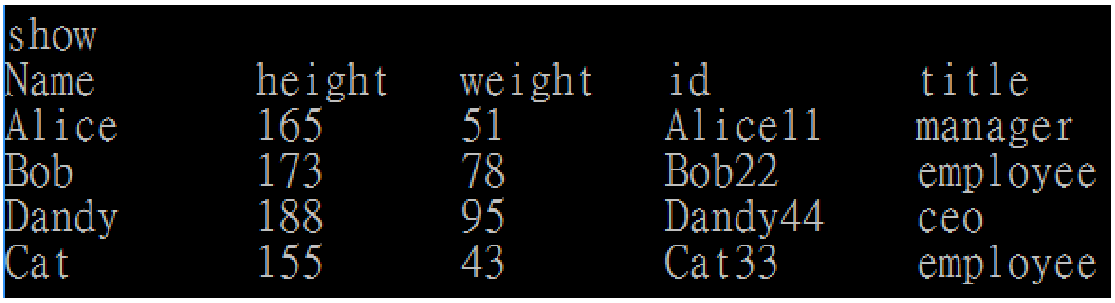
1.



```
assign
Alice 165 51 Alice11@manager.com
Bob 173 78 Bob22@employee.com

show
Name       height   weight   id        title
Alice      165      51       Alice11   manager
Bob        173      78       Bob22     employee
```

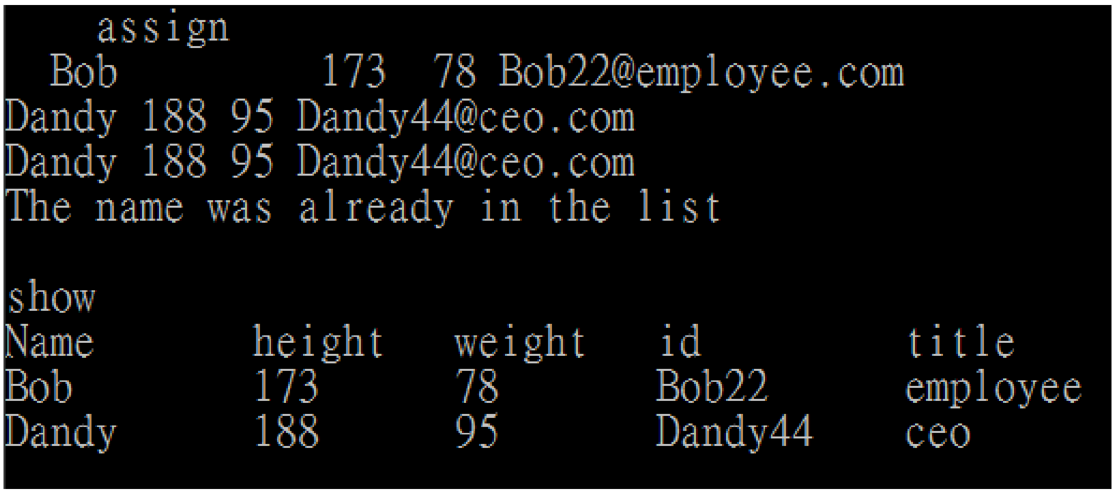


```
assign
Dandy 188 95 Dandy44@ceo.com
Cat 155 43 Cat33@employee.com

show
Name        height   weight   id         title
Alice       165      51       Alice11    manager
Bob         173      78       Bob22      employee
Dandy       188      95       Dandy44    ceo
Cat         155      43       Cat33      employee

sort title
Name        height   weight   id         title
Dandy       188      95       Dandy44    ceo
Alice       165      51       Alice11    manager
Bob         173      78       Bob22      employee
Cat         155      43       Cat33      employee
```

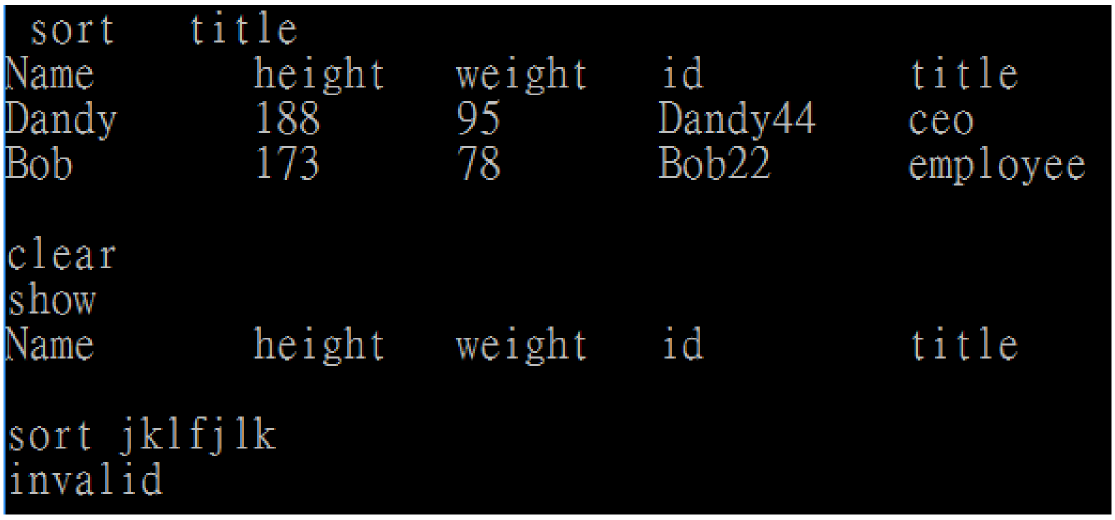


```
show
Name        height   weight   id         title
Alice       165      51       Alice11    manager
Bob         173      78       Bob22      employee
Dandy       188      95       Dandy44    ceo
Cat         155      43       Cat33      employee
```

2.



```
    assign
  Bob         173  78 Bob22@employee.com
Dandy 188 95 Dandy44@ceo.com
Dandy 188 95 Dandy44@ceo.com
The name was already in the list

show
Name        height   weight   id         title
Bob         173      78       Bob22      employee
Dandy       188      95       Dandy44    ceo
```



```
 sort   title
Name         height    weight    id          title
Dandy        188       95        Dandy44     ceo
Bob          173       78        Bob22       employee

clear
show
Name         height    weight    id          title

sort jklfjlk
invalid
```

4. (BONUS) Simple Command Parser 2

Please extend last lab command parser to handle bracketed string:

(1) The characters enclosed by a pair of single quotes ('), should be seen as single world.

(2) In order to input single quotes ('), user should input two single quotes ('').

(3) There are no single quotes in the command head and preposition.

for example:

input: view 'this is ''an'' apple' output: "view" is a type 1 command With 1 argument: "this is 'an' apple"	input: create 'apple, banana', 'orange' as 'fruit,table' output: "create" is a type 2 command with preposition: "as" 1 argument list: "apple, banana" "orange" and 1 tail argument: "fruit,table"
input: insert 'cat', '''' , ' ,,,, ' into 'into' output: "insert" is a type 2 command with preposition: "into" 1 argument list: "cat" "'" " ,,,, " and 1 tail argument: "into"	input: remove 'remove', 'from' from 'remove from' output: "remove" is a type 2 command with preposition: "from" 1 argument list: "remove" "from" and 1 tail argument: "remove from"

Still, there are 3 types of command:

	Command head	Arguments	Description
type 0:	**exit**	none	When user input this command, exit the program.
type 1:	**view**	1	Show its argument when user input this type command. (Please refer to example.)
	delete	1	
type 2	**create**	i1,i2,i3…**as** tailArg	Show its argument when user input this type command. (Please refer to example.)
	insert	i1,i2,i3…**into** tailArg	
	remove	i1,i2,i3…**from** tailArg	

Note:

(1) Your program need to repeat until user input exit.

(2) The space between command and argument **is valid**, and user can also add space before command.

(3) You have to remove all the **space** in argument and them print them.

(4) When your program gets an unknown command such as “xxx 1234”,
it should print “Unknown command "xxx".” And you don’t need to handle other invalid input.