Outline:

1 Requirement

1.0 hello-message and prompt symbol (e.g., >>> $)

You can customize your hello-message when your shell starts running.
The messages will not affect your score.

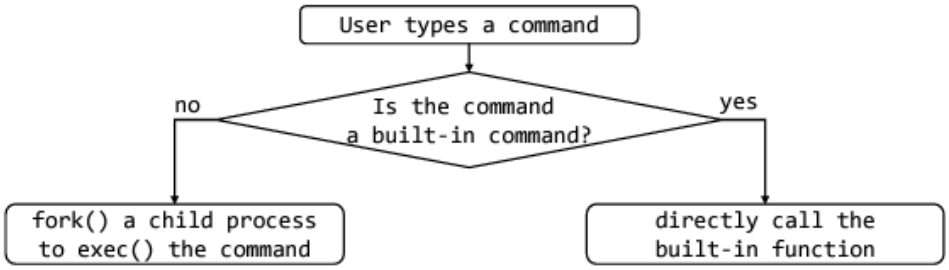
You must print prompt symbol.

```
crlin@crlin-PC:~/桌面/os_hw1_2022/new_sh$ ./my_shell
================================================
* Wellcome to my little shell:                 *
*                                              *
* Type "help" to see builtin functions.        *
*                                              *
* If you want to do things below:              *
* + redirection: ">" or "<"                    *
* + pipe: "|"                                  *
* + background: "&"                            *
* Make sure they are seperated by "(space)".   *
*                                              *
* Have fun!!                                   *
================================================
>>> $
```

1 Requirement

1.1 run built-in command

concept flow-chart: ↓



Please refer to 2 Built-in command requirement
for detail implementation requirements

1 Requirement

1.2 run single process command

If user input a line with only “space“ or “\t” characters,
you should do nothing, and print another prompt symbol.

```
>>> $ ls
text2.txt  text3.txt  text.txt
>>> $ cat text.txt
I study in NCKU
I am a junior student
I take the OS course this semester
I am going to do the os hw1 right away
>>> $
```

```
>>> $
>>> $
>>> $
>>> $
```

1 Requirement

1.3 run two-process pipelines

```
>>> $ cat text.txt
I study in NCKU
I am a junior student
I take the OS course this semester
I am going to do the os hw1 right away
>>> $ cat text.txt | head -1
I study in NCKU
>>> $ cat text.txt | tail -2
I take the OS course this semester
I am going to do the os hw1 right away
>>> $
```

1 Requirement

1.4 handle input and output redirection

```
>>> $ ls
text2.txt  text3.txt  text.txt
```

```
>>> $ cat < text.txt
I study in NCKU
I am a junior student
I take the OS course this semester
I am going to do the os hw1 right away
>>> $ cat text.txt > out_test.txt
>>> $ cat out_test.txt
I study in NCKU
I am a junior student
I take the OS course this semester
I am going to do the os hw1 right away
>>> $
```

1 Requirement

1.5 execute commands in the background

The parent-process (which runs the shell) should print the pid of the child-process (that runs the command in the background).

```
>>> $ ls &
[Pid]: 4897
>>> $ out_test.txt  pipeout.txt  text2.txt  text3.txt  text.txt
```

1 Requirement

1.6 run multi-pipelines

With all the functionalities mentioned before,
your shell should run the command smoothly.

When run multi-pipelines in the background, the original shell process should print the process' pid of the right most command.
(For example, in the screenshot below,
you will print the pid of the process that runs "grep" command.)

(將 multi-pipelines 放到背景執行時, 原 process 要輸出的 pid 為: 執行最右側的指令的 process 的 pid)

```
>>> $ cat < text.txt | head -4 | tail -3 | grep os > pipeout.txt &
[Pid]: 4245
>>> $ cat < text.txt
I study in NCKU
I am a junior student
I take the OS course this semester
I am going to do the os hw1 right away
>>> $ cat pipeout.txt
I am going to do the os hw1 right away
>>> $
```

2 Built-in command requirement

2.1 help/cd/echo/exit

NAME help - print information to stdout
SYNOPSIS help
DESCRIPTION You should at least print how to use the built-in functions.
EXAMPLE

```
>>> $ help
-----------------------------------------------------
my little shell
Type program names and arguments, and hit enter.

The following are built in:
1: help:    show all build-in function info
2: cd:      change directory
3: echo:    echo the strings to standard output
4: record: show last-16 cmds you typed in
5: replay:  re-execute the cmd showed in record
6: mypid:    find and print process-ids
7: exit:    exit shell

Use the "man" command for information on other programs.
-----------------------------------------------------
>>> $
```

2 Built-in command requirement

2.1 help/cd/echo/exit

NAME cd – change the working directory
SYNOPSIS cd [directory]
EXAMPLE

```
>>> $ pwd
/home/crlin/桌面/os_hw1_2022/new_sh/test_dir
>>> $ cd ..
>>> $ pwd
/home/crlin/桌面/os_hw1_2022/new_sh
>>> $ cd test_dir
>>> $ pwd
/home/crlin/桌面/os_hw1_2022/new_sh/test_dir
>>> $
```

2 Built-in command requirement

2.1 help/cd/echo/exit

NAME echo – print a line of text to stdout
SYNOPSIS echo [-n] [strings]
DESCRIPTION If “-n” flag is set, “echo” will not output the trailing newline.
EXAMPLE

```
>>> $ echo 123 456
123 456
>>> $ echo -n 123 456
123 456>>> $ echo -n -n -n
-n -n>>> $ echo enough
enough
>>> $
```

2 Built-in command requirement

2.1 help/cd/echo/exit

NAME exit – terminate your shell
SYNOPSIS exit
DESCRIPTION This command will not run in the backgound.
You may print some goodbye-message before terminate.
EXAMPLE

2 Built-in command requirement

2.2 record/replay

NAME record – show the last-16 commands
SYNOPSIS record
DESCRIPTION Your shell will always record the last-16 commands that user used in the shell. When user type the "record" command, the shell will print the last-16 commands to stdout, including "record" itself. The biggest number indicate the latest command being used (i.e., "record" itself).

If the command is not a legal command (e.g., "recorf" in p.17), that command will still be recorded. The only exception is the "replay" command, which itself will not be recorded.

See next page for example.

2 Built-in command requirement

2.2 record/replay

NAME record – show the last-16 commands

EXAMPLE

```
>>> $ record
history cmd:
 1: echo 100
 2: echo 200
 3: echo 300
 4: echo 400
 5: echo 500
 6: echo 600
 7: echo 700
 8: echo 800
 9: echo 900
10: echo 1000
11: record
12: ls
13: pwd
14: whoami
15: clear
16: record
```

```
>>> $ echo hi
hi
>>> $ record
history cmd:
 1: echo 300
 2: echo 400
 3: echo 500
 4: echo 600
 5: echo 700
 6: echo 800
 7: echo 900
 8: echo 1000
 9: record
10: ls
11: pwd
12: whoami
13: clear
14: record
15: echo hi
16: record

>>> $
```


2 Built-in command requirement

2.2 record/replay

NAME replay – re-execute the command that is listed in record

SYNOPSIS replay [number] (1 <= number <= 16)

DESCRIPTION User should use the “replay” command with a number.
If the number is in legal range, the shell should re-execute the command according to the number listed in the “record” command.

If “replay” is used with wrong argument (not a legal number), your shell should output an error message “replay: wrong args”, and should not execute any command, neither be recorded.

IMPORTANT: The “replay” command itself will not be recorded in the shell. Instead, the command that are actually “be replayed” are recorded in the shell.

See next page for example.

2 Built-in command requirement

2.2 record/replay

NAME replay – re-execute the command listed in record
EXAMPLE

```
>>> $ record
history cmd:
 1: echo 1000
 2: whoami
 3: record
 4: echo 12345
 5: clear
 6: record
 7: echo 1000
 8: clear
 9: recorf
10: record
11: clear
12: record
13: echo 1000
14: clear
15: history
16: record
```

```
>>> $ replay 4
12345
>>> $ record
history cmd:
 1: record
 2: echo 12345
 3: clear
 4: record
 5: echo 1000
 6: clear
 7: recorf
 8: record
 9: clear
10: record
11: echo 1000
12: clear
13: history
14: record
15: echo 12345
16: record

>>> $
```

2 Built-in command requirement

2.3 mypid

NAME mypid – show the related pids about the process
SYNOPSIS mypid [-i|-p|-c] [number] (number indicate a process' pid)
DESCRIPTION Depend on the flag used with the command,
mypid will list the related process' pid(s).

-i: print process' pid, which execute "mypid". (ignore [number])
-p: print process' parent's pid (i.e., who has child [number])
-c: print process' child's pid (i.e., whose parent is [number])

You must implement this command through parsing information in the /proc directory. (except for implementing the "-i" option).

See next two pages for examples and hints.

2 Built-in command requirement

2.3 mypid

NAME mypid - show the related pids about the process

EXAMPLE

```
>>> $ mypid -i
2335
>>> $ mypid -p 2000
mypid -p: process id not exist
>>> $ mypid -p 2335
2248
>>> $ mypid -c 2248
2335
>>> $ mypid -p 2248
2239
>>> $ mypid -c 2239
2248
>>> $ mypid -p 2239
1656
>>> $ mypid -c 1656
1657
1687
1689
1693
1708
1713
1720
```

2 Built-in command requirement

gentle reminder: use fopen() with "r"
or open() with O_RDONLY

2.3 mypid

HINT

The /proc file system will create a directory for each process, using its Pid as its directory name.

```
crlin@crlin-PC:~/桌面/os_hw1_2022/new_sh$ ls /proc/
1     111   169   1877  2034  2290  302   334  471  69   99          locks
10    1110  17    189   2036  2296  303   335  48   7    acpi        mdstat
100   1118  170   1895  2039  2297  3049  338  49   70   asound      meminfo
```

There are many files in the corresponding directory recording the information related to that process (e.g., stat, status, ...).

```
crlin@crlin-PC:~/桌面/os_hw1_2022/new_sh$ ls /proc/2758
arch_status         cwd         mem           patch_state   stat
attr                environ     mountinfo     personality   statm
autogroup           exe         mounts        projid_map    status
auxv                fd          mountstats    root          syscall
```

3 Input format

1. Only 4 special operators: |, >, < and & .
   - No quotation marks(" or '), e.g., "string", 'string'
2. All the cmds, args, operators will be separated by space char.
   - 指令(cmd), 引數(arg) , 特殊符號(operators) 都會用 空白符號 隔開
3. Input/ redirection (<) only show up after first command.
   - Input redirection 的檔名一定會接在 < 後面，且如果有，一定會緊接在第一個指令後面
4. Output redirection (>) only show up after last command.
   - Output redirection 的檔名一定會接在 > 後面，且如果有，一定會緊接在最後一個指令後面
5. Background-execution operator (&) will only show up at last.
   - & 如果有，一定會出現在最後面

格式	`$ cmd args < infile | cmd args | cmd args > outfile &`
範例1	`$ cat < t1.txt > t2.txt &`
範例2	`$ record | head -c 32 > t2.txt &`
範例3	`$ cat < t1.txt | head -5 | tail -3 | grep pid > t2.txt &`

5 Precautions/Reference

- You should implement HW1 with **C** language.
- You will get two files: makefile, my_shell.c
  from the hw1 github classroom.
- Make sure your main() function is written in the file my_shell.c.
- You can modify makefile as you want.
  E.g., add other files and compile them with your my_shell.c
  using your modified makefile.
- Make sure your makefile can compile your codes and create the executable smoothly .
  The executable name should be: my_shell.