HW1 Kernel Module

105072123 黃海茵

- Output

```
[  841.915244] 105072111, 1-9-1997.
[  841.915244] 105072222, 2-10-1997.
[  841.915244] 105072333, 3-11-1997.
[  841.915245] 105072444, 4-12-1997.
[  841.915245] 105072555, 5-1-1998.
```

- Code

```c
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/list.h>
#include <linux/slab.h>

struct data {
        int id;
        int year;
        int month;
        int day;
        struct list_head list;
};

struct data *setting (int id, int year, int month, int day){
        struct data *student = kmalloc(sizeof(struct data), GFP_KERNEL);
        student->id = id;
        student->year = year;
        student->month = month;
        student->day = day;

        return student;
}

static LIST_HEAD(student_list);
```

首先 include 所有會用到的 header 檔，然後建立一個 struct data，包含 id, year, month, day，以及用來儲存 prev 和 next 的 list。function setting 用來配置核心空間記憶體及設定學生資料。最後初始化一個 head 節點 student_list，也就是將 prev 與 next 指標指向自己。

```c
/* This function is called when the module is loaded. */
int simple_init(void)
{
        printk(KERN_INFO "Loading Module\n");

        struct data *student = setting(105072111, 1997, 9, 1);
        list_add_tail(&student->list, &student_list);
        student = setting(105072222, 1997, 10, 2);
        list_add_tail(&student->list, &student_list);
        student = setting(105072333, 1997, 11, 3);
        list_add_tail(&student->list, &student_list);
        student = setting(105072444, 1997, 12, 4);
        list_add_tail(&student->list, &student_list);
        student = setting(105072555, 1998, 1, 5);
        list_add_tail(&student->list, &student_list);

        struct data *ptr;
        list_for_each_entry(ptr, &student_list, list) {
                printk(KERN_INFO "%d, %d-%d-%d.\n", ptr->id, ptr->day, ptr->month, ptr->year);
        }

        return 0;
}
```

function simple_init 中，呼叫 setting 替每位學生設定資料，並使用 list_and_tail 來讓新節點加入 student_list 的尾端，並設為 head 的 prev。然後再用 list_for_each_entry 來 traverse student_list，並依序印出每位學生的資料。

```c
/* This function is called when the module is removed. */
void simple_exit(void)
{
        struct data *ptr, *next;

        list_for_each_entry_safe(ptr, next, &student_list, list){
                list_del(&(ptr->list));
                kfree(ptr);
        }

        printk(KERN_INFO "Removing Module\n");
}
```

fuction simple_exit 中，使用 list_for_each_entry_safe 來 traverse student_list，並將各個 delete 且釋放核心空間記憶體。相較於 list_for_each_entry，list_for_each_entry_safe 用了 pointer next 來儲存 linked list 的下一個節點。所以在 traverse linked list 時如果要 delete 當下的節點，用 list_for_each_entry_safe 可以安全的 delete，不會影響到接下來的 traverse。