雙向鍊結串列庫

libdouble_linked_list.a

使用說明書

李奕承 611121212

使用前的準備

1. #include”dllSpec.h”

2. 編譯時需要加上 -L <你放 libdouble_linked_list.a 的路徑> -l double_linked_list 這兩個參數

3. dllNode_t 結構可以放在新結構中的任意成員順序

```
typedef struct test{
    char data;
    dllNode_t node;
}testNode;
```

當要使用本程式庫提供的函數時，用 <&新結構.dllNode_t 的成員名稱> 來做為參數，以下面這行測試程式為例

```
DLL_addto_next(&a.node, &z.node);
```

4. 使用下面的宏，可以利用 dllNode_t 指針反推出新結構所代表的指針

```
return_to_user_struct_pointer(USER_STRUCT, MEMBER_NAME, MEMBER_POINT)
```

下面是測試程式中的例子，當我們得到一個 dllNode_t 指針，使用宏來得到新結構的指針，就可以存取新結構中的 data 值了

```
int printf_node_data(dllNode_t *current)
{
printf("%c\n", return_to_user_struct_pointer(testNode, node, current)->data);

}
```

自由決定 dllNode_t 在新結構位置順序的原理

考量到一個新結構中可能會有多種不同的子結構，不可能讓所有子結構都當新結構的第一個成員，因此我們需要設計一個機制來存取不同位置的子結構，以下是我設計的宏

```
return_to_user_struct_pointer(USER_STRUCT, MEMBER_NAME, MEMBER_POINT)
```

它利用老師上課提到的結構成員間格(offset)的原理來實現，利用下面這個宏來取得 dllNode_t 成員在新結構中的偏移量

```
#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *)0)->MEMBER)
```

再把 dllNode_t 成員的地址減去偏移量，用強制類型轉換，將新的地址轉為新結構的指針，就可以存取新結構中的其他成員了

```
#define return_to_user_struct_pointer(USER_STRUCT, MEMBER_NAME, MEMBER_POINT)
((USER_STRUCT *)((size_t)MEMBER_POINT - offsetof(USER_STRUCT, MEMBER_NAME)))
```

程式庫函數清單

建構一個空的 list (返回 dllNode_型態的指標)

```
dllNode_t * DLL_init();
```

判斷 head 是否為空的 list

```
int DLL_isEmpty(const dllNode_t *head);
```

得到下一個節點的指標

```
dllNode_t * DLL_next_node(const dllNode_t * node);
```

得到上一個節點的指標

```
dllNode_t * DLL_prev_node(const dllNode_t * node);)
```

計算 List 中有幾個節點

```
unsigned int DLL_num_nodes(const dllNode_t *head);
```

將新節點加入到 List 的第一個

```
void DLL_add_first(dllNode_t * new_node, dllNode_t * head);
```

將新節點加入到 List 的最後一個

```
void DLL_add_tail(dllNode_t * new_node, dllNode_t *head);
```

將新節點加入到節點的前一個

```
void DLL_addto_prev(dllNode_t *new_node, dllNode_t *node);
```

將新節點加入到節點的後一個

```
void DLL_addto_next(dllNode_t *new_node, dllNode_t *node);
```

從節點所在的 Linked List 中刪除此節點

```
void DLL_delete(dllNode_t * node);
```

將 srcList 串在 dstList 之後

```
dllNode_t * DLL_concate(dllNode_t *srcList, dllNode_t * dstList);
```

取得 list 的尾巴

```
dllNode_t *DLL_get_tail(dllNode_t *head);
```

釋放空的 head 指針，並將指針內部的值都 memset 為 0

```
 void DLL_free_head(dllNode_t *head);
```

程式庫宏清單

取得成員在結構中的偏移量

```
offsetof(TYPE, MEMBER)
```

用成員指針反推出結構指針

```
return_to_user_struct_pointer(USER_STRUCT, MEMBER_NAME, MEMBER_POINT)
```

問題回答

- delete node是否需要 free, 請好好去思考。
  - allocate memory 是不是這library需要負責的?
  - free memory 呢?
  - 以上答案 是 Yes/No? Why?

1.no

Delete node 不需要 free，因為當 node 從串列中被刪除時，有可能用來插入在其他串列上，所以不能預設 delete 時一定要跟著 free

2.no

在這個庫中，只負責 list head 指標的 allocate memory，dllNode_t 做為新結構的成員，會跟著使用者設計的新結構一起被 malloc，所以不需要由這個庫來做

3. no

在這個庫中，只負責 list head 指標的 free，dllNode_t 做為新結構的成員，會跟著使用者設計的新結構一起被 free，所以不需要由這個庫來做