雙向鏈結串列庫

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);

1. 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 在新結構成員順序的原理

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。而本結構庫所提供的函數返回的都是dllNode\_t結構的指標，因此我們需要設計一個機制來用返回的dllNode\_t指標回到新結構的指標，這樣才能存取新結構的其他成員，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標，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

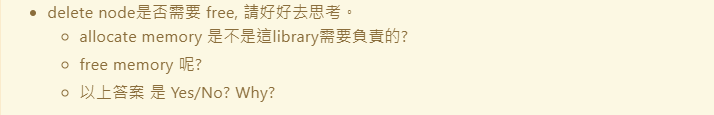
取得成員在結構中的偏移量

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



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，而且node包含在新結構中，當新結構被使用者free時也會一起被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