HW3 Readme

3-1 思路

- 這一題主要是藉由 **singly linked list** 方式，讓資料的增加刪除能夠相較於傳統的**array**更加方便。
- 此題主要是由4個函式組成，接下來我會分別說明。

CODE 分析

MyInsertFunction

```
void MyInsertFunction( int pos , node* first , char* name )
{
    if( pos >  lastpos  || pos < 0 )
    {
        printf("no result\n") ;
        return ;
    }
    else
    {
        node* pos_ptr = first ;
        int cnt = pos ;
        while( cnt-- )
        {
            pos_ptr = pos_ptr->link ;
        }
        node* pos_after_ptr = pos_ptr -> link ;
        // pointer after pos
        if( pos_after_ptr == NULL )
        {
            node* tmp = (node*) malloc(sizeof(node)) ;
            tmp -> link = NULL ;
            pos_ptr->link = tmp ;
            strcpy( tmp ->  data , name ) ;
            lastpos++ ;
        } // pos is the last node int the linked list
        else
        {
            node* tmp = (node*) malloc(sizeof(node)) ;
            tmp->link = pos_after_ptr ;
            pos_ptr-> link = tmp ;
            strcpy( tmp ->  data , name ) ;
```

```
                strcpy( tmp -> data , name ) ;
                lastpos++ ;
            }// pos is not the last node in the linked list
        }
    }
    # insert 我分為兩種case，一種是last node 一種不是。
    # 如果插入的 node 是last node ，他的 link 就是NULL。
```

MyDeleteFunction

```
1  void MyDeleteFunction( int pos , node* first )
2  {
3
4      if( pos > lastpos || pos <= 0 )
5      {
6          printf("no result\n") ;
7          return ;
8      }
9      else
10     {
11         int cnt = 0 ;
12         node* tmp = first ;
13         node* pos_before = first ;
14         node* pos_cur =first ;
15
16         while( cnt <= pos )
17         {
18             if( cnt == pos - 1 )
19             {
20                 pos_before = tmp ;
21             }
22             if( cnt == pos )
23             {
24                 pos_cur = tmp ;
25             }
26             tmp = tmp -> link ;
27             cnt++ ;
28         }
29         // find pointer before delete node to link
30         node* pos_after = pos_cur -> link ;
31         pos_before -> link = pos_after ;
```

```
            pos_before -> link = pos_after ;
            pos_cur->link = NULL ;
            lastpos -- ;
            free( pos_cur)  ;
        }
    }
    # 因為這題有initial node的關係，
      使得delete fuction 不用考慮刪除頭的case。
    # 刪除就是把要刪節點前一個link接到後一個link。
```

MyQueryFunction

```
void MyQueryFunction( int pos , node* first )
{
    node* ans = first ;
    char ans_name[ 30 ] ;
    if( pos > lastpos || pos == 0 )
    {
        printf("no result\n") ;
    }
    else
    {
        while( pos -- )
        {
            ans = ans -> link ;
        }
        strcpy( ans_name , ans->data  ) ;
        printf("%s\n",ans_name) ;
    }
}
#  這邊主要是做linked list 的 search功能。
#  linked list 沒辦法和 array一樣，直接取值。
   需要一個link一個link慢慢接著找。
```

MySwapFuction

```
void MySwapFunction(  int posa , int posb ,
                      node* first )
{
    if( posa <= 0 || posb <= 0 ||
        posa > lastpos || posb > lastpos  )
    {
        printf("no result\n") ;
        return ;
    }// can not swap case
    node*  nodea = first ;
    node*  nodeb = first ;
    node*  preva = NULL ;
    // the node before nodea
    node*  prevb = NULL  ;
    // the node before nodeb
    node* tmp = first ;
    int cnt = 0 ;
    const int maxPos = (posa > posb) ? posa : posb;
    while(  ( tmp != NULL )  && ( cnt <= maxPos ) )
    {
        if ( cnt == posa - 1)
            preva= tmp;
        if ( cnt == posa )
            nodea = tmp;
        if ( cnt == posb - 1)
            prevb = tmp;
        if ( cnt  == posb)
            nodeb = tmp;
        tmp = tmp->link;
        cnt++;
    }// find pointer to get link for swapping
```

```
    }// find pointer to get link for swapping
    if (nodea != NULL && nodeb != NULL)
    {
        if (preva != NULL)
        {
            preva->link= nodeb;
        }
        if (prevb != NULL)
        {
             prevb->link = nodea;
        }
        node* temp  = nodea->link;
        // in case of disappearing when swap
        nodea->link  = nodeb->link;
        nodeb->link = temp;
    }//swap two node
}
# swap fuction 的部分我有另外宣告preva和prevb。

#   這樣做link的時候比較不會搞亂，
    不知道哪個link是從哪裡來的。

#   這邊有另外判斷 NULL 的問題
    避免指到不該只到地方造成run error。
```

3-2 思路

- 這一題主要是藉由circular linked list 的方式 ， 重複刪除，直到刪到最後一個之後停止。

- 我的作法是先依照順逆時鐘去做 **first node link**的**方向**，再寫一個刪除的function。用一個while迴圈去判斷說**剩下一個node**就結束刪除。

CODE 分析

輸入處理函式

- 輸入處理的部分，我用了strcut先暫存名字。
  等到知道是**順還是逆時鐘**我再把這先name放好。

```
struct people
{
    char name[ 20 ] ;
}peoples[ 1000 ]; // record name
```

```
void ProcessInput( char* list_name )
{
    int len = strlen( list_name ) ;
    int name_idx = 0 ;
    for( int i = 0 ; i < len-1 ; i++ )
    {
        if( list_name[ i ] == ',' )
        {
            num_people++ ;
            name_idx = 0 ;
        }
        else if( list_name[ i ] == ' ' )
        {
            continue ;
        }
        else
        {
            peoples[ num_people ].name[ name_idx ]
            = list_name[ i ] ;
            name_idx++ ;
        }// PUT name in struct for copy to pointer
    }
}

# 首先我用了fgets先讀取第一行，因為同時有空白和逗號。
# 接著再用for掃一次，依序再strcut people放入名字。

```

CreateList 函式

- 得知時鐘方向開始接link。

```
void Clockwise( int k , node* first )
{
    int cnt = num_people ;
    int people_idx = 1 ;
    // peoples_index in struct
    node* cur_ptr = first ;
    // current pointer is first
    while( cnt-- )
    {
        node* tmp = ( node* )malloc( sizeof( node ) ) ;
        cur_ptr->link = tmp ;
        cur_ptr = tmp  ;
        tmp->link = NULL ;
        strcpy(    tmp -> data ,
                   peoples[ people_idx ].name ) ;
        people_idx++ ;
    }
    cur_ptr->link = first ;
    // circular linked list need
    // link last node to head

}
void CounterClockwise( int k , node* first )
{
    int cnt = num_people ;
    int people_idx = num_people ;
    // peoples_index in struct
    node* cur_ptr = first ;
    // current pointer is first
    while( cnt-- )
    {
```

```
32          node* tmp = ( node* )malloc( sizeof( node ) ) ;
33          cur_ptr->link = tmp ;
34          cur_ptr = tmp  ;
35          tmp->link = NULL ;
36          strcpy(  tmp -> data ,
37                   peoples[ people_idx ].name ) ;
38          people_idx-- ;
39      }
40      cur_ptr->link = first ;
41      // circular linked list need
42      // link last node to head
43  }
44  # 這邊就是create linked list的部分。
45    很單純，每個插入的node都是最後一個。
46
47  # 需要注意的部分是，因為是circular linked list，
48    所以需要將last node 接到 first node 身上。
```

刪除函式

- 終止條件

```
while( num_survive >= 2 )
{
    start = FindSurviver( k , start ,first ) ;
    num_survive-- ;
}
while( num_survive >= 2 )
{
    start = FindSurviver( k , start ,first ) ;
    num_survive--;
}
```

- FindSurviver Function

```
node* FindSurviver( int k , node* start ,
                            node* first )
{

    int count = 0 ;
    node* flag = start ;
    node* cur = start ;
    node* prev = start ;
    while (  prev->link != start )
    {
        prev = prev->link;
        count++;
    }
    // find the node before delete node to link
    while( k-- )
    {
        prev = cur ;
        cur = cur->link ;
    }
    // find the node before delete node to link
    printf("%s is killed.\n",cur->data) ;
    prev->link= cur->link ;
    if( cur == first )
    {
        first = cur->link  ;
    }
    // DELETE first node case
    flag = cur->link ;
    free( cur ) ;
    // check pointer in case of link to NULL
    if( prev != NULL )
```

```
31          if( prev != NULL )
32          {
33              cur = prev->link ;
34          }
35          else
36          {
37              cur = NULL ;
38          }
39          return flag ;
40          // flag for the start for next deletion
41      }
42      # circular linked list 刪除時注意是否刪除為第一個node。
43        如果是沒更動first的地址，會指到 NULL 造成run error。
```