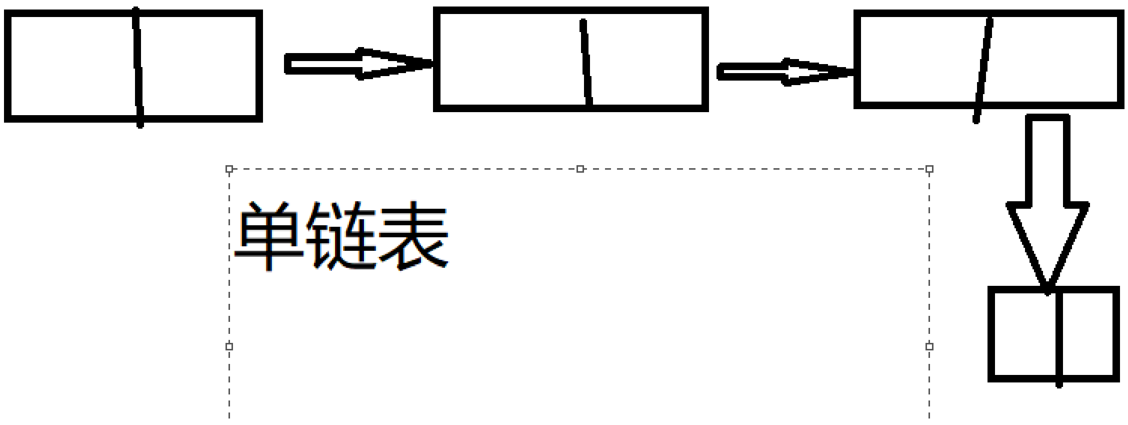
C语言经典单链表API详解

数据结构一览

具体的源码位置在这里！ ： click here

这一次，我们开发的是链表这个数据结构，基础的链表分为单链表，双链表和循环链表。作为链表，他不同之处在于，我们的数据存储是离散的！就是说，我们的一串数据不一定是在内存上连续，我们的逻辑结构依靠指针来维护：



可以看到，我们使用指针来访问数据！为了方便我们的维护，那当然是一个抽象成一个链子串起了若干的节点，这就在直观上需要两个数据结构：

```
typedef struct _DataNode_{
    void* data;              // 数据域
    struct _DataNode_* pNext;// 指针域
}DataNode;
```

而在一般的教程中是如下定义的：

```
typedef struct _DataNode_{
    void* data;              // 数据域
    struct _DataNode_* pNext;// 指针域
}DataNode,*Datalist;
```

这里我并不这样做，这是为了可读性与方便维护的优点。为了增添属性，我们以加入该链表存在的元素个数作为附属属性：

```
typedef struct _DataList_{
    DataNode* head;
    size_t current_num;
}DataList;
```

于是，现在我们开始维护与开发相关的操作函数！

链表权威导论

链表是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。链表由一系列结点（链表中每一个元素称为结点）组成，结点可以在运行时动态生成。**每个结点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针域。** 相比于线性表顺序结构，操作复杂。**由于不必须按顺序存储，链表在插入的时候可以达到O(1)的复杂度，比另一种线性表顺序表快得多，但是查找一个节点或者访问特定编号的节点则需要O(n)的时间，而线性表和顺序表相应的时间复杂度分别是O(logn)和O(1)。**

使用链表结构可以克服数组链表需要预先知道数据大小的缺点，**链表结构可以充分利用计算机内存空间，实现灵活的内存动态管理。但是链表失去了数组随机读取的优点，同时链表由于增加了结点的指针域，空间开销比较大。链表最明显的好处就是，常规数组排列关联项目的方式可能不同于这些数据项目在记忆体或磁盘上顺序，数据的存取往往要在不同的排列顺序中转换。链表允许插入和移除表上任意位置上的节点，但是不允许随机存取。**

基本导入

1）预先功能启用与宏定义

```
#define _CRT_SECURE_NO_WARNINGS 1 // 取消非安全警报
#define OPENUPPOSARR 1  // 启用动态位置查询存储数组
#define OPENUPDEFAULTSIZE 1 // 启用默认快捷大小宏
#define OPENQUICKMALLOC 1 // 启用快捷开辟堆内存宏
```

2）头文件使用

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
```

3）错误码一览

```
//Error Code Meaning
//Program Normally run till the end return NORMAL as 0
//When Error in mallocing Space the program return 1
//when inputing a NULL we returns -1
//when inputing a invalid input we returns -2
//Locations UnFind we returns -3
typedef enum _ClassicLinkListFunctionStatues_ {
    ClassicLinkList_NORMAL=0,
    ClassicLinkList_ERROR_IN_MALLOCING_SPACE = 1,
    ClassicLinkList_NULL_INPUT = -1,
    ClassicLinkList_Invalid_Input=-2,
```

```c
    ClassicLinkList_UnFind=-3,
    ClassicLinkList_Invalid_Free=-4,
}ClassicLinkListFunctionStatues;
```

4）自定义我们的布尔类型

```c
//MyBool Defined here!
//
typedef enum _bool_ {
    True = 1,
    False = 0
}Bool;
```

5）报错机制与参数调整宏

```c
//This is some abstractions using in reminding you some basic informations and errors!
//
#define SHOW_ERROR_ClassicLinkList_NULL_INPUT printf("\nSorry! Your input NULL!\n")
#define SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE printf("\nSorry!Failed to malloc
space for your data\n")
#define SHOW_ERROR_ClassicLinkList_Invalid_Input printf("\nYour input invalid, reject to run
functions\n")
#define SHOW_ERROR_ClassicLinkList_UnFind printf("\nPositions unfind!\n")
#define SHOW_ERROR_ClassicLinkList_Invalid_Free printf("\nFree the invalid space,reject to run
the functions\n")
#define SHOW_WARNING_ClassicLinkList_SWAPPED_POS printf("Warning! the front_pos is bigger then
the final pos\n Do you agreed swap them?")
#define SHOW_WARNING_ClassicLinkList_OVERLAP_POS printf("Considered as your insertion_input is
overlapped!\n set pos as the cur_size?")
//this abstractions is used to define the aquisition of whether the program's decison is
acceptable,
//X is the message string!
//
#define SHOW_WHETHER_ACCCEPTED(X) printf(X)

//If wanted flashed a newline you can use it!
//
#define SLASHN printf("\n")

//used in MergeclassicLinkList
//If accepted copy !  Do write ClassicLinkList_ACCEPTED_COPY
#define ClassicLinkList_ACCEPTED_COPY 1
#define ClassicLinkList_NO_COPY 0
#define Unfind_ClassicLinkList -1
```

6）位置查询结果枚举

```
//Used in whether the targeted elements is found
//If is considered found, just use Find , otherwise unfind!
typedef enum _ClassicLinkList_isFind_ {
    Find = 1,
    Unfind = 0
}ClassicLinkList_isFind;
```

7）更好的打印开启枚举

```
//Used in whether print data in ClassicLinkList better
//Use ClassicLinkList_OPEN if you want a better print
typedef enum _BetterPrintSwitch_for_ClassicLinkList_ {
    ClassicLinkList_OPEN = 1,
    ClassicLinkList_CLOSE = 0
}BetterPrintSwitch_for_ClassicLinkList;
```

8）一些类型的重改写

```
typedef unsigned int Is_Used_Compared_Int;
typedef void(*MyPrint)(void*);
typedef void* (*Do_Specific_Change)(void*);
typedef Is_Used_Compared_Int(*CompareFunc)(void*, void*);
typedef Is_Used_Compared_Int(*LocateFunc)(void*, void*);
```

9）默认大小宏

注意，希望关闭默认大小快捷宏请移除宏

```
#if OPENUPDEFAULTSIZE


#define Dyarr_SIGINT sizeof(int)
#define Dyarr_SIGCHAR sizeof(char)
#define Dyarr_SIGFLOAT sizeof(float)
#define Dyarr_SIGDOUBLE sizeof(double)
#define Dyarr_SIG(dataType) sizeof(dataType)

#define Dyarr_INT_SIZE(X) sizeof(int)*X
#define Dyarr_CHAR_SIZE(X) sizeof(char)*X
#define Dyarr_FLOAT_SIZE(X) sizeof(float)*X
#define Dyarr_DOUBLE_SIZE(X) sizeof(double)*X
#define Dyarr_TYPE_SIZE(dataType,X) sizeof(dataType)*X

#endif
```

10）快速开辟

注意，希望关闭快速开辟宏请移除宏

```
#if OPENQUICKMALLOC

#define MALLOC(type) (type*)malloc(sizeof(type))
#define MALLOCN(type,ElemNum) (type*)malloc(sizeof(type)*ElemNum)

#endif
```

11）动态存储位置

更加详细的API请参考“C语言结构：动态数组API详解”这篇文章

```
//-----------------------Position_Stored_Dynamic_Array_For_Dyarr---------------------------------
-
//About Position_Stored_Dynamic_Array_Func
//Create A Position_Stored_Dynamic_Array! Can init it in this way:
//>>Position_Stored_Dynamic_ArrayFordyarr* Name=Init_A_Postion_Stored_Dynamic_ArrayFordyarr();
//Used in returning multitude locations

#if OPENUPPOSARR

//Main data structure!
//Used in returning multitude locations

typedef struct _Dyarrposarr_ {
    int* posSpace;
    size_t pos_size;
}Position_Stored_Dynamic_ArrayFordyarr;

//Want a better Print to beutify your control table? try this one

typedef enum _BetterPrintSwitch_for_pos_dyarr_ {
    PSDA_dyarr_OPEN = 1,
    PSDA_dyarr_CLOSE = 0
}BetterPrintSwitch_for_pos_dyarr;

//can use this type when indicating the type

typedef unsigned int BetterPrintfor_PosDyarr;

//About Position_Stored_Dynamic_Array_Func
//Create A Position_Stored_Dynamic_Array! Can init it in this way:
//>>Position_Stored_Dynamic_ArrayFordyarr* Name=Init_A_Postion_Stored_Dynamic_ArrayFordyarr();

Position_Stored_Dynamic_ArrayFordyarr* Init_A_Postion_Stored_Dynamic_ArrayFordyarr()
{
```

```c
    Position_Stored_Dynamic_ArrayFordyarr* pro_usable_space =
(Position_Stored_Dynamic_ArrayFordyarr*)malloc(sizeof(Position_Stored_Dynamic_ArrayFordyarr));
    if (!pro_usable_space)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    pro_usable_space->posSpace = NULL;
    pro_usable_space->pos_size = 0;
    return pro_usable_space;
}

//About Positions_Stored_Dynamic_Array
//this function is aimed to output a series of locations that is stored in the pos
////Use PSDA_dyarr_OPEN to have a better shown ,else use PSDA_dyarr_CLOSE

ClassicLinkListFunctionStatues Show_All_Locations_In_PSDAfor_dyarr(
    Position_Stored_Dynamic_ArrayFordyarr*                  posarr,
    BetterPrintfor_PosDyarr                                 Whether_Better_Print
)
{
    if (!posarr)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (Whether_Better_Print == PSDA_dyarr_OPEN)
        printf("\nPositions are shown! :\n");
    for (int i = 0; i < posarr->pos_size; i++)
        printf("%u ", posarr->posSpace[i]);
    if (Whether_Better_Print == PSDA_dyarr_OPEN)
        printf("\nFinish Printing!\n");
    return ClassicLinkList_NORMAL;
}

//About Positions_Stored_Dynamic_Array
ClassicLinkListFunctionStatues Push_back_a_locations_in_PSDA_dyarr(
    Position_Stored_Dynamic_ArrayFordyarr*                  posarr,
    int                                                     pos_acquired
)
{
    int* pro_usable_space = (int*)realloc(posarr->posSpace, sizeof(int) * (posarr->pos_size +
1));
    if (!pro_usable_space)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }
    posarr->posSpace = pro_usable_space;
    posarr->posSpace[posarr->pos_size] = pos_acquired;
    posarr->pos_size++;
    return ClassicLinkList_NORMAL;
}
//About Positions_Stored_Dynamic_Array
//this function is aimed to return the number of locations
//Use PSDA_dyarr_OPEN to have a better shown ,else use PSDA_dyarr_CLOSE
size_t Get_pos_size_From_PSDA_for_dyarr(
    Position_Stored_Dynamic_ArrayFordyarr*                  posarr,
    BetterPrintfor_PosDyarr                                 Whether_Shown_Print
)
{
    if (!posarr)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (Whether_Shown_Print == PSDA_dyarr_OPEN)
        printf("\nCurrent targeted data's total num is:%u!\n", posarr->pos_size);
    return posarr->pos_size;
}

//About Positions_Stored_Dynamic_Array
//return a pos that is targeted!
//
void* getPosbyPosinPSDA(
    Position_Stored_Dynamic_ArrayFordyarr* getter,
    size_t                                      pos
)
{
    if (!getter)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (pos > getter->pos_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    for (int i = 0; i < pos; i++)
        return (char*)getter->posSpace + i * sizeof(size_t);
}

//About Positions_Stored_Dynamic_Array
ClassicLinkListFunctionStatues DesTroy_A_PSDA_for_dyarr(
    Position_Stored_Dynamic_ArrayFordyarr*                  pointer_to_wishedfreeposarr
)
{
    if (!pointer_to_wishedfreeposarr)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    free(pointer_to_wishedfreeposarr->posSpace);
    free(pointer_to_wishedfreeposarr);
    pointer_to_wishedfreeposarr = NULL;
    return ClassicLinkList_NORMAL;
}
#endif //   OPENUPPOSARR
```

核心API 1：工程函数

我们的工厂函数依旧是老三样：默认，拷贝和升级！

这里是源文件文档内的注释：

```
//----------------------------Init-functions---------------------------
//
// there are three basic type of initialization
//
// -------------------------------------------------------------------
// 1.includeing the default type (init a NULL head and zero cur_size)
// func1: Classic_DataList* Init_A_ClassicLinkList()
//
// No input , or input VOID
// have output : output a clssicLinklist pointer
// -------------------------------------------------------------------
// 2.includeing the copy one (init the same type of the be-copied one)
// func2: Classic_DataList* Init_A_ClassicLinkList_By_CopyAClassicLinkList
//
// input : a required_be_copied list .
// output: a new list but have is the same as the be_copied list
// -------------------------------------------------------------------
// 3.includeing to transform the static array to the linklist one
// func3: Classic_DataList* UpdateStaticArray2ClassicLinkList
//
// input : a waited_transformed static_array , the element size which is in the static array
// output: a new pointer that points to classicLinkList
//
//
//----------------------------Init-functions---------------------------
```

默认工厂函数

由于我们定义了一个全新的结构：链表！我们就希望，先什么都不放！换而言之：head是空的，而current_size也是0！同时，使用 malloc 构造出来这个结构并且传出一个指针来供外界托管与使用：

原型这样就出来了：

```
Classic_DataList* Init_A_ClassicLinkList()
```

开辟堆空间：

```
// create empty space
Classic_DataList* pro_usable_space = (Classic_DataList*)malloc(sizeof(Classic_DataList));
```

我们当然最好是检查一下有没有开辟成功

```
// Check whether init succeeded
if (!pro_usable_space)
{
    SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
    exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
}
```

开辟成功了那就自然开始默认设置

```
// Default set
pro_usable_space->Head = NULL;
pro_usable_space->current_size = 0;

// return out the Structure
return pro_usable_space;
```

连起来就是如下：

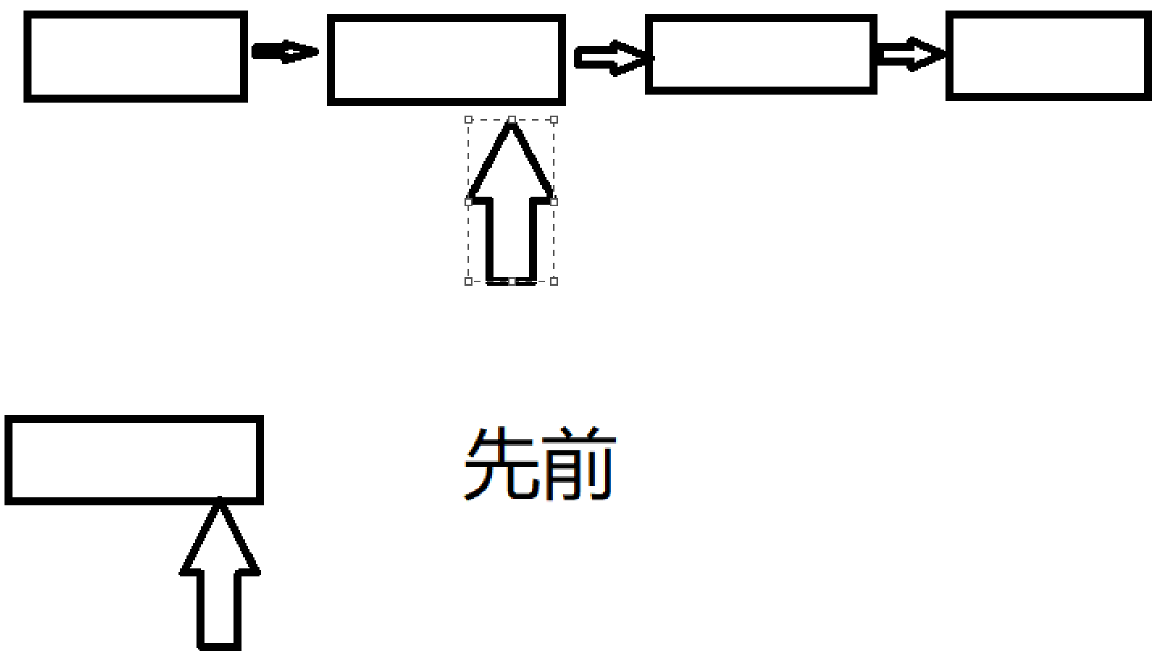
```
Classic_DataList* Init_A_ClassicLinkList()
{
    // create empty space
    Classic_DataList* pro_usable_space = (Classic_DataList*)malloc(sizeof(Classic_DataList));

    // Check whether init succeeded
    if (!pro_usable_space)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }

    // Default set
    pro_usable_space->Head = NULL;
    pro_usable_space->current_size = 0;

    // return out the Structure
    return pro_usable_space;
}
```

拷贝工厂函数

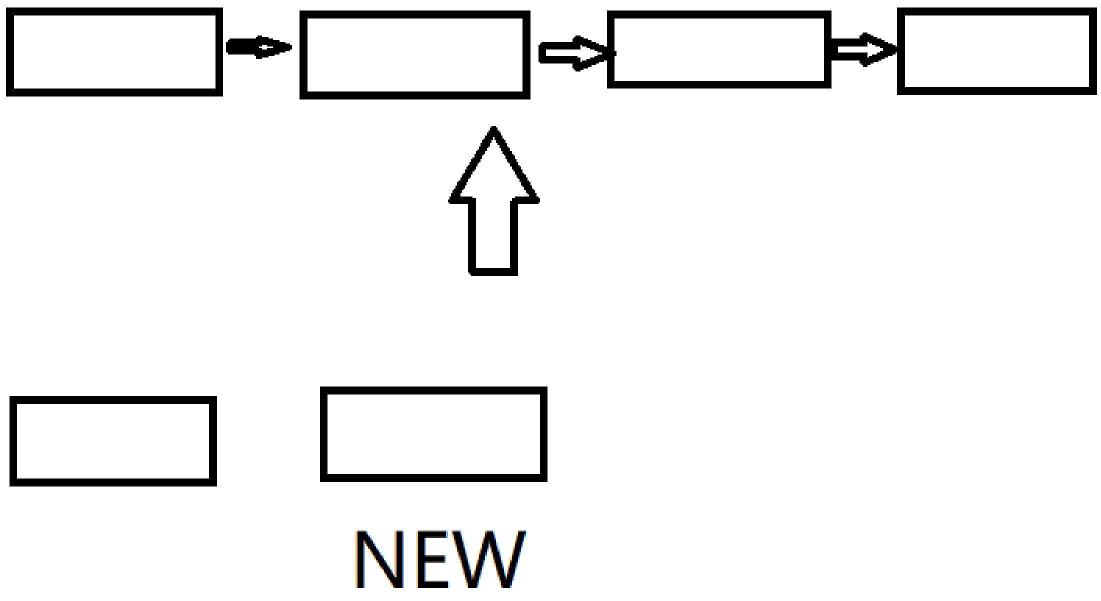
这个函数顾名思义就可以写出原型：

```
Classic_DataList* Init_A_ClassicLinkList_By_CopyAClassicLinkList(
    Classic_DataList*                                        list
)
```

这个就有意思了，我们的想法是简答的，我们选择两个指针从链表里走，什么意思呢？



然后，我们开始拷贝被拷贝链表的结点：



然后，指针连上：顺便进入新的拷贝节点！周而复始直到被拷贝链表指向 NULL

做一点简单判断：让空的或者是非法的指针先排除

```
//check whether the input is legal
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    //check whether there is neccessity to copy
    if (list->current_size == 0)
    {
        printf("No need to copy! it hasn't been initialize yet!");
        //Used to return NULL , but it is dangerous that two pointers points one things
        return NULL;
    }
```

然后，拷贝头节点，做一点简单的初始化（不调用其他工厂函数，尽可能在一个函数完成事情，减少链接丢失带来的失败的风险）！

我们拷贝数据，是要拷贝节点框架+节点数据：

```
//New a List and new a head and new the data of the head
// 产生一个链表
    Classic_DataList* p_NewList = (Classic_DataList*)malloc(sizeof(Classic_DataList));
// 产生一个头节点准备入列
    Classic_DataNode* newHead = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
// 头节点数据拷贝一份
    void* newdata = malloc(sizeof(list->Head->data));
```

数据拷贝检查：

```
//check whether we init the node successfully
    if (newHead == NULL)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }

    if (newdata == NULL)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }
```

数据拷贝

```
    //Do copy the data to the new head and finished initialization
//拷贝节点数据
    memcpy(newdata, list->Head->data, sizeof(list->Head->data));
//指针托管
    newHead->data = newdata;
//防止野指针或者多进程解引用带来野指针访问
    newHead->p_next_one = NULL;
    p_NewList->Head = newHead;
    p_NewList->current_size = 1;
```

这样，我们可以开始拷贝其他的节点了，正如上文所说，准备两个节点指针：

```
    //Start copy other nodes
    Classic_DataNode* p_Currency = list->Head->p_next_one;
    Classic_DataNode* pCurCopy = p_NewList->Head;
```

类似的重复拷贝头节点的工作：

```
    for (int i = 0; i < list->current_size-1; i++) //注意是cur - 1 !
    {
        //Copy the I-th node
        Classic_DataNode* newDataNode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));

        //check if the initialization is illegal
        if(newDataNode==NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        // prepare to copy data from the current node
        void* newdata = malloc(sizeof(p_Currency->data));

        // check if the initialization is illegal
        if (newdata == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        //copy the detailed node
        memcpy(newdata, p_Currency->data, sizeof(p_Currency->data));
        newDataNode->data = newdata;
        newDataNode->p_next_one = NULL;

        //Moving the pointer and prepared next copy
        pCurCopy->p_next_one = newDataNode;
        pCurCopy = pCurCopy->p_next_one;
        p_Currency = p_Currency->p_next_one;
    }
```

链表的属性完善以下，返回指针以便托管

```
//Data copy finished , init the cur_size
    p_NewList->current_size = list->current_size;
    return p_NewList;
```

连起来看一下：

```
// About ClassicLinkList
// to copy an already exited linklist
// Classic_DataList* List = Init_A_ClassicLinkList_By_CopyAClassicLinkList(Classic_DataList*
list)
//
Classic_DataList* Init_A_ClassicLinkList_By_CopyAClassicLinkList(
    Classic_DataList*                                       list
)
```

```
{
    //check whether the input is legal
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    //check whether there is neccessity to copy
    if (list->current_size == 0)
    {
        printf("No need to copy! it hasn't been initialize yet!");
        //Used to return NULL , but it is dangerous that two pointers points one things
        return NULL;
    }

    //New a List and new a head and new the data of the head
    Classic_DataList* p_NewList = (Classic_DataList*)malloc(sizeof(Classic_DataList));
    Classic_DataNode* newHead = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
    void* newdata = malloc(sizeof(list->Head->data));

    //check whether we init the node successfully
    if (newHead == NULL)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }

    if (newdata == NULL)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }

    //Do copy the data to the new head and finished initialization
    memcpy(newdata, list->Head->data, sizeof(list->Head->data));
    newHead->data = newdata;
    newHead->p_next_one = NULL;
    p_NewList->Head = newHead;
    p_NewList->current_size = 1;

    //Start copy other nodes
    Classic_DataNode* p_Currency = list->Head->p_next_one;
    Classic_DataNode* pCurCopy = p_NewList->Head;
    for (int i = 0; i < list->current_size-1; i++)
    {
        //Copy the I-th node
        Classic_DataNode* newDataNode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));

        //check if the initialization is illegal
        if(newDataNode==NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        // prepare to copy data from the current node
        void* newdata = malloc(sizeof(p_Currency->data));
        // check if the initialization is illegal
        if (newdata == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        //copy the detailed node
        memcpy(newdata, p_Currency->data, sizeof(p_Currency->data));
        newDataNode->data = newdata;
        newDataNode->p_next_one = NULL;

        //Moving the pointer and prepared next copy
        pCurCopy->p_next_one = newDataNode;
        pCurCopy = pCurCopy->p_next_one;
        p_Currency = p_Currency->p_next_one;
    }

    //Data copy finished , init the cur_size
    p_NewList->current_size = list->current_size;
    return p_NewList;
}
```

升级工厂函数

现在我们把一个数组的元素拆开，存入链表里，那么，每一个元素都作为一个节点穿进绳子里！于是：

函数首先就需要数组的地址，和元素的大小以及元素个数！

```
Classic_DataList* UpdateStaticArray2ClassicLinkList(
    void*                                       inputArray,
    size_t                                      dataSize,
    size_t                                      dataNum
)
```

接下来，判断是不是空指针：

```
if (inputArray == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
```

初始化头节点：

```
Classic_DataList* pro_usable_space = (Classic_DataList*)malloc(sizeof(Classic_DataList));

Classic_DataNode* FirstHeaddata = (Classic_DataNode*)malloc(sizeof(Classic_DataNode*));

void* datapiece = malloc(dataSize);
if (
    pro_usable_space == NULL
    &&
    FirstHeaddata == NULL
    &&
    datapiece == NULL
    )
{
    SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
    exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
}

memcpy(datapiece, inputArray, dataSize);

FirstHeaddata->data = datapiece;
FirstHeaddata->p_next_one = NULL;

pro_usable_space->current_size = 1;
pro_usable_space->Head = FirstHeaddata;
```

相似的，但是拷贝节点是通过访问数组的地址来拷贝的：

```
for (int i = 1; i < dataNum; i++)
{
    Classic_DataNode* p_Currency = pro_usable_space->Head;
    while (p_Currency->p_next_one)
    {
        p_Currency = p_Currency->p_next_one;
    }
    void* datastored = malloc(dataSize);
    if (!datastored)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }
    memcpy(datastored, (char*)inputArray + i * dataSize, dataSize);
    Classic_DataNode* datanode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
    if (!datanode)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }
    datanode->data = datastored;
    datanode->p_next_one = NULL;
    p_Currency->p_next_one = datanode;
    pro_usable_space->current_size++;
}
return pro_usable_space;
```

工厂函数一览

```
//----------------------------Init-functions----------------------------------------------------
----------------
//
// there are three basic type of initialization
```

```
//
// ---------------------------------------------------------------------------------------------
---------------
// 1.includeing the default type (init a NULL head and zero cur_size)
// func1: Classic_DataList* Init_A_ClassicLinkList()
//
// No input , or input VOID
// have output : output a clssicLinklist pointer
// ---------------------------------------------------------------------------------------------
----------------
// 2.includeing the copy one (init the same type of the be-copied one)
// func2: Classic_DataList* Init_A_ClassicLinkList_By_CopyAClassicLinkList
//
// input : a required_be_copied list .
// output: a new list but have is the same as the be_copied list
// ---------------------------------------------------------------------------------------------
----------------
// 3.includeing to transform the static array to the linklist one
// func3: Classic_DataList* UpdateStaticArray2ClassicLinkList
//
// input : a waited_transformed static_array , the element size which is in the static array
// output: a new pointer that points to classicLinkList
//
//
//----------------------------Init-functions--------------------------------------------------
----------------


// About ClassicLinkList
// This function is used to init a default type of classicLinkList
// that the head of dataPiece is NULL and the cur_size is zero.
// Use push back to init the dataPiece
// use in this Way:
// Classic_DataList* List = Init_A_ClassicLinkList();


Classic_DataList* Init_A_ClassicLinkList()
{
    // create empty space
    Classic_DataList* pro_usable_space = (Classic_DataList*)malloc(sizeof(Classic_DataList));

    // Check whether init succeeded
    if (!pro_usable_space)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }

    // Default set
    pro_usable_space->Head = NULL;
    pro_usable_space->current_size = 0;

    // return out the Structure
    return pro_usable_space;
}

// About ClassicLinkList
// to copy an already exited linklist
// Classic_DataList* List = Init_A_ClassicLinkList_By_CopyAClassicLinkList(Classic_DataList*
list)
//
Classic_DataList* Init_A_ClassicLinkList_By_CopyAClassicLinkList(
    Classic_DataList*                                       list
)
{
    //check whether the input is legal
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    //check whether there is neccessity to copy
    if (list->current_size == 0)
    {
        printf("No need to copy! it hasn't been initialize yet!");
        //Used to return NULL , but it is dangerous that two pointers points one things
        return NULL;
    }

    //New a List and new a head and new the data of the head
    Classic_DataList* p_NewList = (Classic_DataList*)malloc(sizeof(Classic_DataList));
    Classic_DataNode* newHead = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
    void* newdata = malloc(sizeof(list->Head->data));

    //check whether we init the node successfully
    if (newHead == NULL)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }

    if (newdata == NULL)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }

    //Do copy the data to the new head and finished initialization
    memcpy(newdata, list->Head->data, sizeof(list->Head->data));
    newHead->data = newdata;
    newHead->p_next_one = NULL;
    p_NewList->Head = newHead;
    p_NewList->current_size = 1;

    //Start copy other nodes
    Classic_DataNode* p_Currency = list->Head->p_next_one;
    Classic_DataNode* pCurCopy = p_NewList->Head;
    for (int i = 0; i < list->current_size-1; i++)
    {
        //Copy the I-th node
        Classic_DataNode* newDataNode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));

        //check if the initialization is illegal
        if(newDataNode==NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        // prepare to copy data from the current node
        void* newdata = malloc(sizeof(p_Currency->data));

        // check if the initialization is illegal
        if (newdata == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        //copy the detailed node
        memcpy(newdata, p_Currency->data, sizeof(p_Currency->data));
        newDataNode->data = newdata;
        newDataNode->p_next_one = NULL;

        //Moving the pointer and prepared next copy
        pCurCopy->p_next_one = newDataNode;
        pCurCopy = pCurCopy->p_next_one;
        p_Currency = p_Currency->p_next_one;
    }

    //Data copy finished , init the cur_size
    p_NewList->current_size = list->current_size;
    return p_NewList;
}

// About ClassicLinkList
// to updata a static array into a ClassicLinkList
// Classic_DataList* List = UpdateStaticArray2ClassicLinkList(
// void* inputArray,
// size_t dataSize,
// size_t dataNum
// )
//
Classic_DataList* UpdateStaticArray2ClassicLinkList(
    void*                                       inputArray,
    size_t                                      dataSize,
    size_t                                      dataNum
)
{
    if (inputArray == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataList* pro_usable_space = (Classic_DataList*)malloc(sizeof(Classic_DataList));
    Classic_DataNode* FirstHeaddata = (Classic_DataNode*)malloc(sizeof(Classic_DataNode*));
    void* datapiece = malloc(dataSize);
    if (
        pro_usable_space == NULL
        &&
        FirstHeaddata == NULL
        &&
        datapiece == NULL
        )
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }
    memcpy(datapiece, inputArray, dataSize);
    FirstHeaddata->data = datapiece;
    FirstHeaddata->p_next_one = NULL;
    pro_usable_space->current_size = 1;
    pro_usable_space->Head = FirstHeaddata;
    for (int i = 1; i < dataNum; i++)
    {
        Classic_DataNode* p_Currency = pro_usable_space->Head;
        while (p_Currency->p_next_one)
        {
            p_Currency = p_Currency->p_next_one;
        }
        void* datastored = malloc(dataSize);
        if (!datastored)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        memcpy(datastored, (char*)inputArray + i * dataSize, dataSize);
        Classic_DataNode* datanode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        if (!datanode)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        datanode->data = datastored;
        datanode->p_next_one = NULL;
        p_Currency->p_next_one = datanode;
        pro_usable_space->current_size++;
    }
    return pro_usable_space;
}
```

基本添加函数

现在，我们执行向里面添加数据的方法，说白了，就是创造一个新的节点，或者是若干的节点向里面推送数据：

```
//----------------------------------------Basic_Add_Functions----------
// there are four basic input functions:
//
// -------------------------------------------------------------------
// 1. push back an element into the classicLinkList
// func1 : ClassicLinkListFunctionStatues Push_Back_Into_A_ClassicLinkList
//
// input : the inserted list ,the data wanted to be inserted and the inserted datasize
// output: the statues of normal ,can be used in check or just ignore!
//
// -------------------------------------------------------------------
// 2. insert back an element into the classicLinkList by offering a valid pos
// func2 : ClassicLinkListFunctionStatues Insert_into_AClassicLinkList
//
//
// input : the inserted list , the data ,the datasize as well as the position you wanted to
insert
// output: the statues of normal ,can be used in check or just ignore!
//
// -------------------------------------------------------------------
//
// 3. push back a bunch of data into the classicLinkList like the first function
// func3 :ClassicLinkListFunctionStatues push_Back_BunchDataintoClassicLinkList
//
// input : the inserted list , the data ,the datasize as well as the position you wanted to
insert
// output: the statues of normal ,can be used in check or just ignore!
//
// -------------------------------------------------------------------
// 4. insert back a bunch of data into the classicLinkList like the second function
// func4: ClassicLinkListFunctionStatues Insert_A_Bunch_of_data_intoClassicLinkList
//
// input : the inserted list , the static array ,the datasize in the static array as well as the
position you wanted to insert
// output: the statues of normal ,can be used in check or just ignore!
//
//-------------------------Basic_Add_Functions-------------------------
```

尾插一个节点

我们首先得拿到一个尾节点，由于是单链表，我们就必须遍历链表到最后面。然后再伸出指针拷贝！函数的原型显而易见，就不再重复了

```
ClassicLinkListFunctionStatues Push_Back_Into_A_ClassicLinkList(
    Classic_DataList*                                    list,
    void*                                               data,
    size_t                                              datasize
)
```

但是这里注意，我们需要区分一个事情，如果我们是创建了一个空链表，那就我们直接推进头节点就好了，于是：

```
void* datastored = malloc(datasize);
if (!datastored)
{
    SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
    exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
}
memcpy(datastored, data, datasize);
Classic_DataNode* datanode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
if (!datanode)
{
    SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
    exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
}
```

```
datanode->data = datastored;
datanode->p_next_one = NULL;
list->Head = datanode;
list->current_size++;
```

如果不是，就要一个一个跑路：

```
//移动指针到尾部
while (p_Currency->p_next_one)
{
    p_Currency = p_Currency->p_next_one;
}

//准备拷贝
void* datastored = malloc(datasize);
if (!datastored)
{
    SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
    exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
}
memcpy(datastored, data, datasize);
Classic_DataNode* datanode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
if (!datanode)
{
    SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
    exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
}
datanode->data = datastored;
datanode->p_next_one = NULL;
p_Currency->p_next_one = datanode;
list->current_size++;
```

连起来看看：

```
// About ClassicLinkList
// push back some data ,like dynamic array , into the final position
//Used like this Push_Back_Into_A_ClassicLinkList(appendedClassicLinkList, data ,sizeof(data))
//
ClassicLinkListFunctionStatues Push_Back_Into_A_ClassicLinkList(
    Classic_DataList*                                           list,
    void*                                                       data,
    size_t                                                      datasize
)
{
    if (!list)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (!data)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* p_Currency = list->Head;
    if(list->Head != NULL)
    {
        while (p_Currency->p_next_one)
        {
            p_Currency = p_Currency->p_next_one;
        }
        void* datastored = malloc(datasize);
        if (!datastored)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        memcpy(datastored, data, datasize);
        Classic_DataNode* datanode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        if (!datanode)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        datanode->data = datastored;
        datanode->p_next_one = NULL;
        p_Currency->p_next_one = datanode;
        list->current_size++;
    }
    else
    {
        void* datastored = malloc(datasize);
        if (!datastored)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        memcpy(datastored, data, datasize);
        Classic_DataNode* datanode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        if (!datanode)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        datanode->data = datastored;
        datanode->p_next_one = NULL;
        list->Head = datanode;
        list->current_size++;
    }
    return ClassicLinkList_NORMAL;
}
```

尾插多个节点

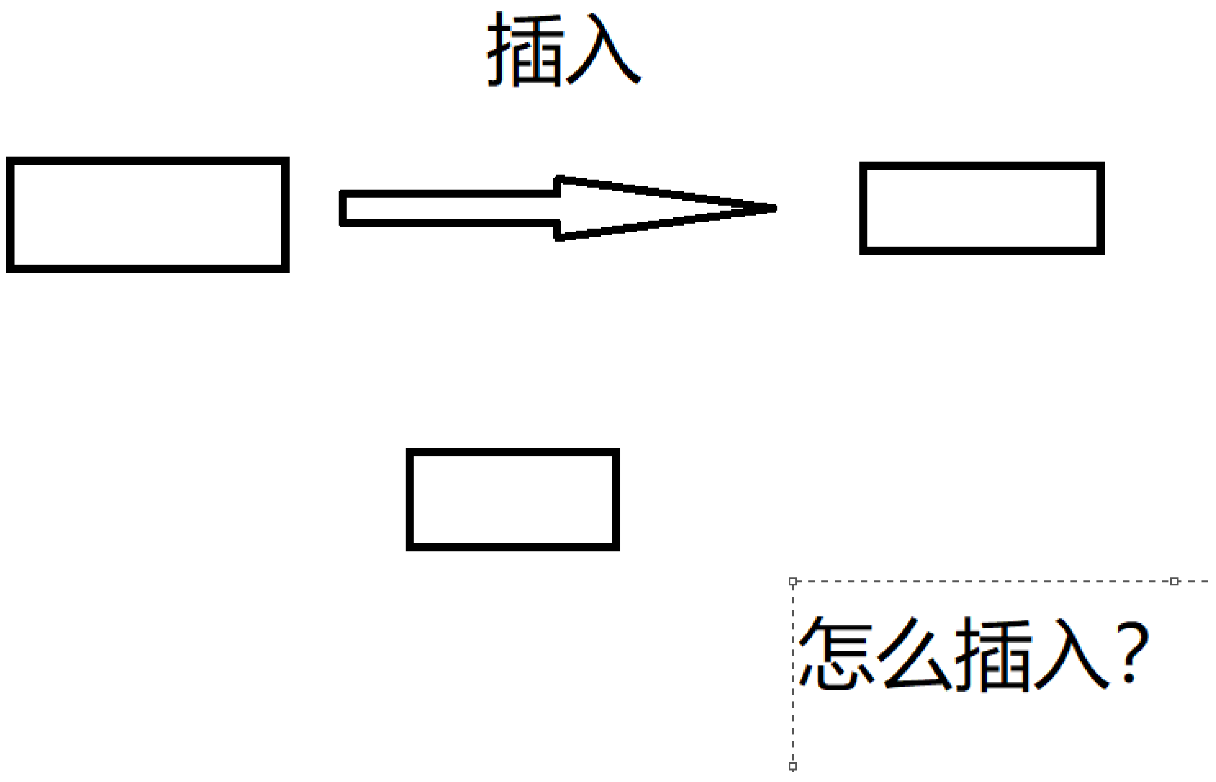
这个很简单，由于是以数组的形式，那我们就仿照升级函数，不过，还是要注意是不是为空链表的问题，总而言之，不细讲，看代码！

```
// About ClassicDataList
// push back a bunch data into the datalist
// used like this : push_Back_BunchDataintoClassicLinkList( inserted_list , the static array
,the elementsize ,the element num)
```

```
//
ClassicLinkListFunctionStatues push_Back_BunchDataintoClassicLinkList(
    Classic_DataList*                                       list,
    void*                                                   inputArray,
    size_t                                                  sigElemSize,
    size_t                                                  arrayNum
)
{

    //数据检查
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }

    if(inputArray==NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }


    //Equally as to updata a Static Array into a ClassicLinkList
    //Once just use  UpdateStaticArray2ClassicLinkList(),but i'm afraid of unable to use this
functions
    //as it is said: "DO NOT motivate another functions , try to finish all work if neccessary"

    //这是经典的升级工厂函数
    if (list->Head == NULL)
    {
        //New a List and new a head and new the data of the head
        Classic_DataList* p_NewList = (Classic_DataList*)malloc(sizeof(Classic_DataList));
        Classic_DataNode* newHead = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        void* newdata = malloc(sizeof(list->Head->data));

        //check whether we init the node successfully
        if (newHead == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        if (newdata == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        //Do copy the data to the new head and finished initialization
        memcpy(newdata, list->Head->data, sizeof(list->Head->data));
        newHead->data = newdata;
        newHead->p_next_one = NULL;
        p_NewList->Head = newHead;
        p_NewList->current_size = 1;

        //Start copy other nodes
        Classic_DataNode* p_Currency = list->Head->p_next_one;
        Classic_DataNode* pCurCopy = p_NewList->Head;
        for (int i = 0; i < list->current_size - 1; i++)
        {
            //Copy the I-th node
            Classic_DataNode* newDataNode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));

            //check if the initialization is illegal
            if (newDataNode == NULL)
            {
                SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
                exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
            }

            // prepare to copy data from the current node
            void* newdata = malloc(sizeof(p_Currency->data));

            // check if the initialization is illegal
            if (newdata == NULL)
            {
                SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
                exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
            }

            //copy the detailed node
            memcpy(newdata, p_Currency->data, sizeof(p_Currency->data));
            newDataNode->data = newdata;
            newDataNode->p_next_one = NULL;

            //Moving the pointer and prepared next copy
            pCurCopy->p_next_one = newDataNode;
            pCurCopy = pCurCopy->p_next_one;
            p_Currency = p_Currency->p_next_one;
        }

        //Data copy finished , init the cur_size
        p_NewList->current_size = list->current_size;
        list = p_NewList;
        return ClassicLinkList_NORMAL;
    }

    //如果不是，那说明可以准备尾插
    //for this section , that means the head isn't NULL , just copy as usual:
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < list->current_size-1; i++)
    {
        pCur = pCur->p_next_one;
    }

    for (int i = 0; i < arrayNum; i++)
    {
        Classic_DataNode* pCopyCur = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        void* copyData = malloc(sizeof(sigElemSize));
        if (pCopyCur == NULL && copyData == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        memcpy(copyData, (char*)inputArray + i * sigElemSize, sigElemSize);
        pCopyCur->data = copyData;
        pCopyCur->p_next_one = NULL;
        pCur->p_next_one = pCopyCur;
        pCur = pCur->p_next_one;
    }

    list->current_size += arrayNum;
    return ClassicLinkList_NORMAL;
}
```

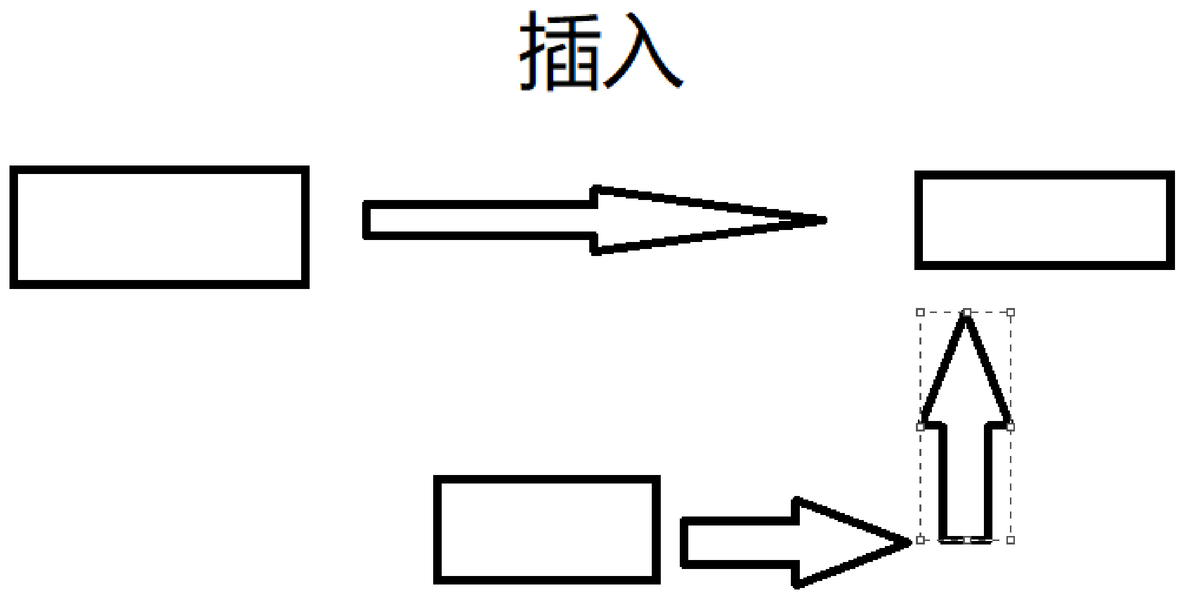
插入一个节点

欸！这里就到了链表的一个关键了！

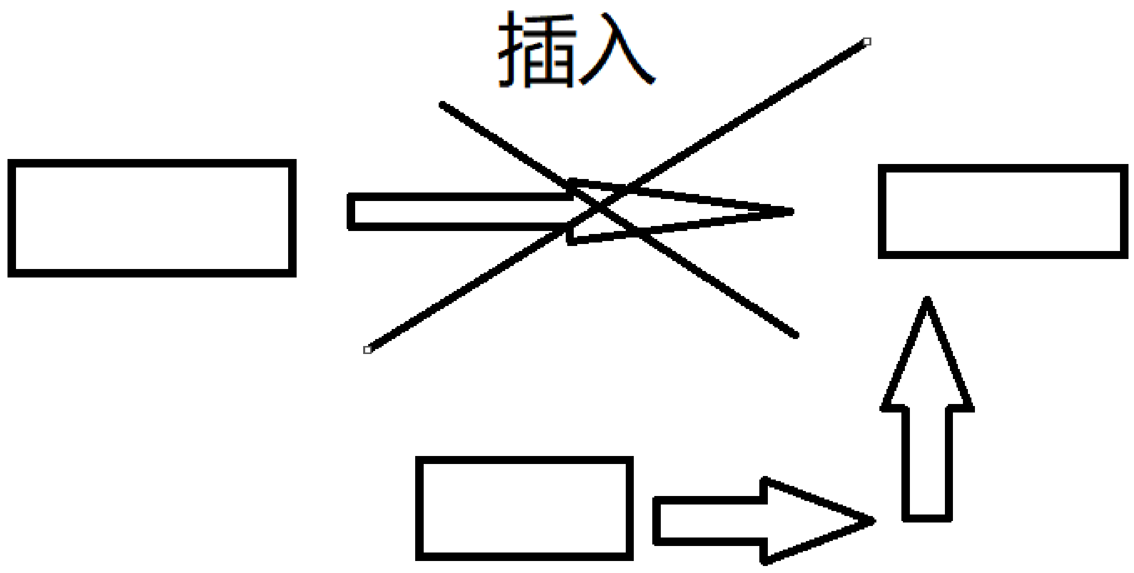


还是要注意到，如果我们断开指针，这样子重连就好了，但是注意，我们必须**先让新节点连上下一个节点，然后旧的链接断开，再连上前一个**，原因很容易猜到的：先断开了怎么找下一个呢？

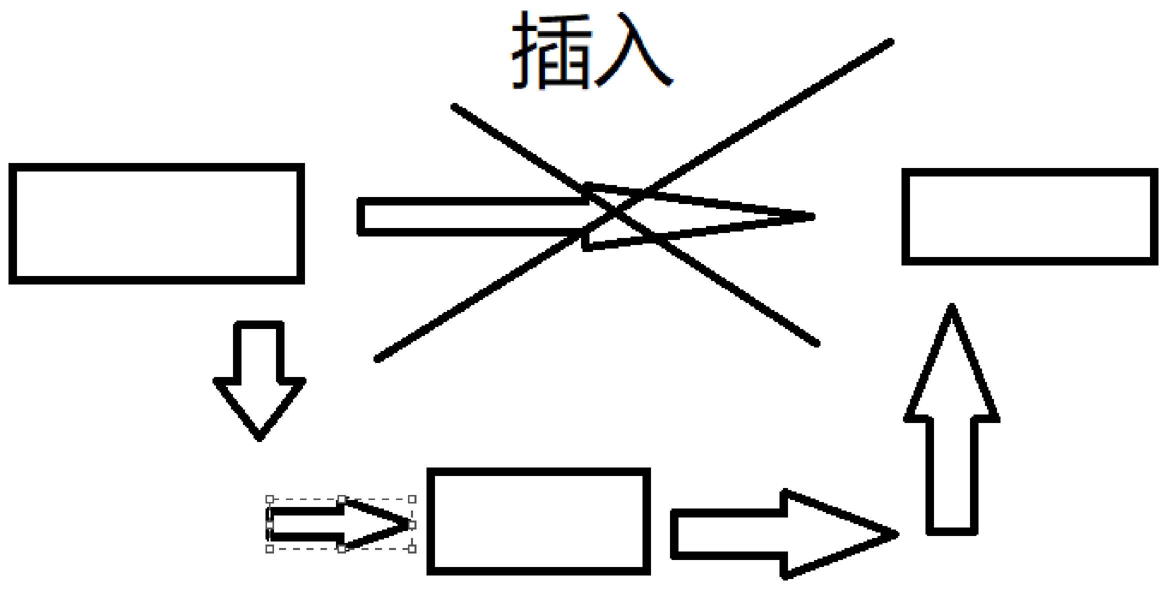
1）



2)



3)



来看看代码：

我们的函数需要知道元素的大小和地址，和插入的位置：

```
ClassicLinkListFunctionStatues Insert_into_AClassicLinkList(
    Classic_DataList*                                        list,
    void*                                                    data,
    size_t                                                   dataSize,
    size_t                                                   pos
)
```

检查数据：

```
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (data == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (pos<0 || pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
```

依旧区分问题：我们插在哪里呢？如果插在头节点，那就：让节点指向旧的头节点，然后把新节点作为头节点就好了

```
    if (pos == 0)
    {
        Classic_DataNode* stored = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        if (stored == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        void* dataPiece = malloc(dataSize);
        memcpy(dataPiece, data, dataSize);
        stored->data = dataPiece;
        stored->p_next_one = list->Head;
        list->Head = stored;
        list->current_size++;
        return ClassicLinkList_NORMAL;
    }
```

中间位置的就把我们的思路抽象出来：

```
    //准备空间
    Classic_DataNode* pCurrency = list->Head;
    Classic_DataNode* stored = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
    if (stored == NULL)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }
    void* dataPiece = malloc(dataSize);
    memcpy(dataPiece, data, dataSize);
    stored->data = dataPiece;
//定位到当前位置
    for (int i = 0; i < pos; i++)
    {
        pCurrency = pCurrency->p_next_one;
    }
//开始入列
    stored->p_next_one = pCurrency->p_next_one;
    pCurrency->p_next_one = stored;
//处理属性
    list->current_size++;
    return ClassicLinkList_NORMAL;
```

就好了！

```
// About ClassicDataList
// insert back a data into the datalist
// used like this: Insert_into_AClassicLinkList(inserted list, input
data,dataSize,insertion_pos)
//
ClassicLinkListFunctionStatues Insert_into_AClassicLinkList(
    Classic_DataList*                                        list,
    void*                                                    data,
    size_t                                                   dataSize,
    size_t                                                   pos
)
{

    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
```

```
            exit(ClassicLinkList_NULL_INPUT);
        }
        if (data == NULL)
        {
            SHOW_ERROR_ClassicLinkList_NULL_INPUT;
            exit(ClassicLinkList_NULL_INPUT);
        }
        if (pos<0 || pos>list->current_size)
        {
            SHOW_ERROR_ClassicLinkList_Invalid_Input;
            exit(ClassicLinkList_Invalid_Input);
        }
        if (pos == 0)
        {
            Classic_DataNode* stored = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
            if (stored == NULL)
            {
                SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
                exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
            }
            void* dataPiece = malloc(dataSize);
            memcpy(dataPiece, data, dataSize);
            stored->data = dataPiece;
            stored->p_next_one = list->Head;
            list->Head = stored;
            list->current_size++;
            return ClassicLinkList_NORMAL;
        }
        Classic_DataNode* pCurrency = list->Head;
        Classic_DataNode* stored = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        if (stored == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        void* dataPiece = malloc(dataSize);
        memcpy(dataPiece, data, dataSize);
        stored->data = dataPiece;
        for (int i = 0; i < pos; i++)
        {
            pCurrency = pCurrency->p_next_one;
        }
        stored->p_next_one = pCurrency->p_next_one;
        pCurrency->p_next_one = stored;
        list->current_size++;
        return ClassicLinkList_NORMAL;
    }
```

插入一群节点函数

很是类似，我们先把目标数组升级，然后找到保存头尾节点，头节点连在插入节点，尾节点连接在下一节点就好了：

函数原型是不难设想的：

```
// About ClassicDataList
// insert back a bunch of data into the datalist
// used like this: Insert_into_AClassicLinkList(inserted list, insertion_pos,input array ,Single
dataSize, the amount of elements)
//
ClassicLinkListFunctionStatues Insert_A_Bunch_of_data_intoClassicLinkList(
    Classic_DataList*                                  list,
    size_t                                            pos,
    void*                                             inputArray,
    size_t                                            sigElemSize,
    size_t                                            arrayNum
)
```

接下来是：

数据判断：

```

    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (inputArray == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    // 询问用户要不要更改pos
    if (pos > list->current_size)
    {
        SHOW_WARNING_ClassicLinkList_OVERLAP_POS;
        SHOW_WHETHER_ACCCEPTED("y/n: y for yes , n for n");
        int choice;
        scanf("%c", &choice);
        if (choice == 'y')
        {
            pos = list->current_size;
        }
        else
        {
            SHOW_ERROR_ClassicLinkList_Invalid_Input;
            exit(ClassicLinkList_Invalid_Input);
        }
    }
```

先升级目标数组

```
    Classic_DataNode* pCurNow = NULL;
    Classic_DataNode* pStart = NULL;

    for(int i = 0 ; i < arrayNum ; i++)
    {

        Classic_DataNode* pCopyCur =    (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        void* copydata = malloc(sigElemSize);
        if (pCopyCur == NULL && copydata == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        memcpy(copydata, (char*)inputArray + i * sigElemSize,           sigElemSize);
        pCopyCur->data = copydata;
        pCopyCur->p_next_one = NULL;

        if (i == 0)
        {
            pCurNow = pCopyCur;
            pStart = pCopyCur;
            continue;
        }
        pCurNow->p_next_one = pCopyCur;
        pCurNow = pCurNow->p_next_one;
    }
```

然后，开始分类讨论

头节点的：

```
if (pos == 0)
    {
     // 转移
        pCurNow->p_next_one = list->Head;
     // 托管新的头节点
        list->Head = pStart;
        list->current_size += arrayNum;
        return ClassicLinkList_NORMAL;
    }
```

也有不是的，那就：

```
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < pos; i++)
    {
        pCur = pCur->p_next_one;
    }

    pCurNow->p_next_one = pCur->p_next_one;
    pCur->p_next_one = pStart;
    list->current_size += arrayNum;
    return ClassicLinkList_NORMAL;
```

增加函数一览

```
//----------------------------------------Basic_Add_Functions------------------------
// there are four basic input functions:
//
// ----------------------------------------------------------------------------------
// 1. push back an element into the classicLinkList
// func1 : ClassicLinkListFunctionStatues Push_Back_Into_A_ClassicLinkList
//
// input : the inserted list ,the data wanted to be inserted and the inserted datasize
// output: the statues of normal ,can be used in check or just ignore!
//
// ----------------------------------------------------------------------------------
// 2. insert back an element into the classicLinkList by offering a valid pos
// func2 : ClassicLinkListFunctionStatues Insert_into_AClassicLinkList
//
//
// input : the inserted list , the data ,the datasize as well as the position you wanted to
insert
// output: the statues of normal ,can be used in check or just ignore!
//
// ----------------------------------------------------------------------------------
//
// 3. push back a bunch of data into the classicLinkList like the first function
// func3 :ClassicLinkListFunctionStatues push_Back_BunchDataintoClassicLinkList
//
// input : the inserted list , the data ,the datasize as well as the position you wanted to
insert
// output: the statues of normal ,can be used in check or just ignore!
//
// ----------------------------------------------------------------------------------
// 4. insert back a bunch of data into the classicLinkList like the second function
// func4: ClassicLinkListFunctionStatues Insert_A_Bunch_of_data_intoClassicLinkList
//
// input : the inserted list , the static array ,the datasize in the static array as well as the
position you wanted to insert
// output: the statues of normal ,can be used in check or just ignore!
//
//----------------------------------------Basic_Add_Functions-------------------------

// About ClassicLinkList
// push back some data ,like dynamic array , into the final position
//Used like this Push_Back_Into_A_ClassicLinkList(appendedClassicLinkList, data ,sizeof(data))
//
ClassicLinkListFunctionStatues Push_Back_Into_A_ClassicLinkList(
    Classic_DataList*                                           list,
    void*                                                       data,
    size_t                                                      datasize
)
{
    if (!list)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (!data)
    {
```

```c
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* p_Currency = list->Head;
    if(list->Head != NULL)
    {
        while (p_Currency->p_next_one)
        {
            p_Currency = p_Currency->p_next_one;
        }
        void* datastored = malloc(datasize);
        if (!datastored)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        memcpy(datastored, data, datasize);
        Classic_DataNode* datanode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        if (!datanode)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        datanode->data = datastored;
        datanode->p_next_one = NULL;
        p_Currency->p_next_one = datanode;
        list->current_size++;
    }
    else
    {
        void* datastored = malloc(datasize);
        if (!datastored)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        memcpy(datastored, data, datasize);
        Classic_DataNode* datanode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        if (!datanode)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        datanode->data = datastored;
        datanode->p_next_one = NULL;
        list->Head = datanode;
        list->current_size++;
    }
    return ClassicLinkList_NORMAL;
}

// About ClassicDataList
// insert back a data into the datalist
// used like this: Insert_into_AClassicLinkList(inserted list, input
data,dataSize,insertion_pos)
//
ClassicLinkListFunctionStatues Insert_into_AClassicLinkList(
    Classic_DataList*                                       list,
    void*                                                   data,
    size_t                                                  dataSize,
    size_t                                                  pos
)
{

    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (data == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (pos<0 || pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (pos == 0)
    {
        Classic_DataNode* stored = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        if (stored == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        void* dataPiece = malloc(dataSize);
        memcpy(dataPiece, data, dataSize);
        stored->data = dataPiece;
        stored->p_next_one = list->Head;
        list->Head = stored;
        list->current_size++;
        return ClassicLinkList_NORMAL;
    }
    Classic_DataNode* pCurrency = list->Head;
    Classic_DataNode* stored = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
    if (stored == NULL)
    {
        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
    }
    void* dataPiece = malloc(dataSize);
    memcpy(dataPiece, data, dataSize);
    stored->data = dataPiece;
    for (int i = 0; i < pos; i++)
    {
        pCurrency = pCurrency->p_next_one;
    }
    stored->p_next_one = pCurrency->p_next_one;
    pCurrency->p_next_one = stored;
    list->current_size++;
    return ClassicLinkList_NORMAL;
}

// About ClassicDataList
// push back a bunch data into the datalist
// used like this : push_Back_BunchDataintoClassicLinkList( inserted_list , the static array
,the elementsize ,the element num)
//
ClassicLinkListFunctionStatues push_Back_BunchDataintoClassicLinkList(
    Classic_DataList*                                        list,
    void*                                                    inputArray,
    size_t                                                   sigElemSize,
    size_t                                                   arrayNum
)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }

    if(inputArray==NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }


    //Equally as to updata a Static Array into a ClassicLinkList
    //Once just use  UpdateStaticArray2ClassicLinkList(),but i'm afraid of unable to use this
functions
    //as it is said: "DO NOT motivate another functions , try to finish all work if neccessary"
    if (list->Head == NULL)
    {
        //New a List and new a head and new the data of the head
        Classic_DataList* p_NewList = (Classic_DataList*)malloc(sizeof(Classic_DataList));
        Classic_DataNode* newHead = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        void* newdata = malloc(sizeof(list->Head->data));

        //check whether we init the node successfully
        if (newHead == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        if (newdata == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        //Do copy the data to the new head and finished initialization
        memcpy(newdata, list->Head->data, sizeof(list->Head->data));
        newHead->data = newdata;
        newHead->p_next_one = NULL;
        p_NewList->Head = newHead;
        p_NewList->current_size = 1;

        //Start copy other nodes
        Classic_DataNode* p_Currency = list->Head->p_next_one;
        Classic_DataNode* pCurCopy = p_NewList->Head;
        for (int i = 0; i < list->current_size - 1; i++)
        {
            //Copy the I-th node
            Classic_DataNode* newDataNode = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));

            //check if the initialization is illegal
            if (newDataNode == NULL)
            {
                SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
                exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
            }

            // prepare to copy data from the current node
            void* newdata = malloc(sizeof(p_Currency->data));

            // check if the initialization is illegal
            if (newdata == NULL)
            {
                SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
                exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
            }

            //copy the detailed node
            memcpy(newdata, p_Currency->data, sizeof(p_Currency->data));
            newDataNode->data = newdata;
            newDataNode->p_next_one = NULL;

            //Moving the pointer and prepared next copy
            pCurCopy->p_next_one = newDataNode;
            pCurCopy = pCurCopy->p_next_one;
            p_Currency = p_Currency->p_next_one;
        }

        //Data copy finished , init the cur_size
        p_NewList->current_size = list->current_size;
        list = p_NewList;
        return ClassicLinkList_NORMAL;
    }

    //for this section , that means the head isn't NULL , just copy as usual:
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < list->current_size-1; i++)
    {
        pCur = pCur->p_next_one;
    }

    for (int i = 0; i < arrayNum; i++)
    {
        Classic_DataNode* pCopyCur = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        void* copyData = malloc(sizeof(sigElemSize));
        if (pCopyCur == NULL && copyData == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        memcpy(copyData, (char*)inputArray + i * sigElemSize, sigElemSize);
        pCopyCur->data = copyData;
        pCopyCur->p_next_one = NULL;
        pCur->p_next_one = pCopyCur;
        pCur = pCur->p_next_one;
    }

    list->current_size += arrayNum;
    return ClassicLinkList_NORMAL;
}

// About ClassicDataList
// insert back a bunch of data into the datalist
// used like this: Insert_into_AClassicLinkList(inserted list, insertion_pos,input array ,Single
dataSize, the amount of elements)
//
ClassicLinkListFunctionStatues Insert_A_Bunch_of_data_intoClassicLinkList(
    Classic_DataList*                               list,
    size_t                                          pos,
    void*                                           inputArray,
    size_t                                          sigElemSize,
    size_t                                          arrayNum
)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (inputArray == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }

    if (pos > list->current_size)
    {
        SHOW_WARNING_ClassicLinkList_OVERLAP_POS;
        SHOW_WHETHER_ACCCEPTED("y/n: y for yes , n for n");
        int choice;
        scanf("%c", &choice);
        if (choice == 'y')
        {
            pos = list->current_size;
        }
        else
        {
            SHOW_ERROR_ClassicLinkList_Invalid_Input;
            exit(ClassicLinkList_Invalid_Input);
        }
    }

    Classic_DataNode* pCurNow = NULL;
    Classic_DataNode* pStart = NULL;

    for(int i = 0 ; i < arrayNum ; i++)
    {

        Classic_DataNode* pCopyCur = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        void* copydata = malloc(sigElemSize);
```

```c
        if (pCopyCur == NULL && copydata == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }

        memcpy(copydata, (char*)inputArray + i * sigElemSize, sigElemSize);
        pCopyCur->data = copydata;
        pCopyCur->p_next_one = NULL;

        if (i == 0)
        {
            pCurNow = pCopyCur;
            pStart = pCopyCur;
            continue;
        }
        pCurNow->p_next_one = pCopyCur;
        pCurNow = pCurNow->p_next_one;
    }

    if (pos == 0)
    {
        pCurNow->p_next_one = list->Head;
        list->Head = pStart;
        list->current_size += arrayNum;
        return ClassicLinkList_NORMAL;
    }

    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < pos; i++)
    {
        pCur = pCur->p_next_one;
    }

    pCurNow->p_next_one = pCur->p_next_one;
    pCur->p_next_one = pStart;
    list->current_size += arrayNum;
    return ClassicLinkList_NORMAL;
}
```

删除函数

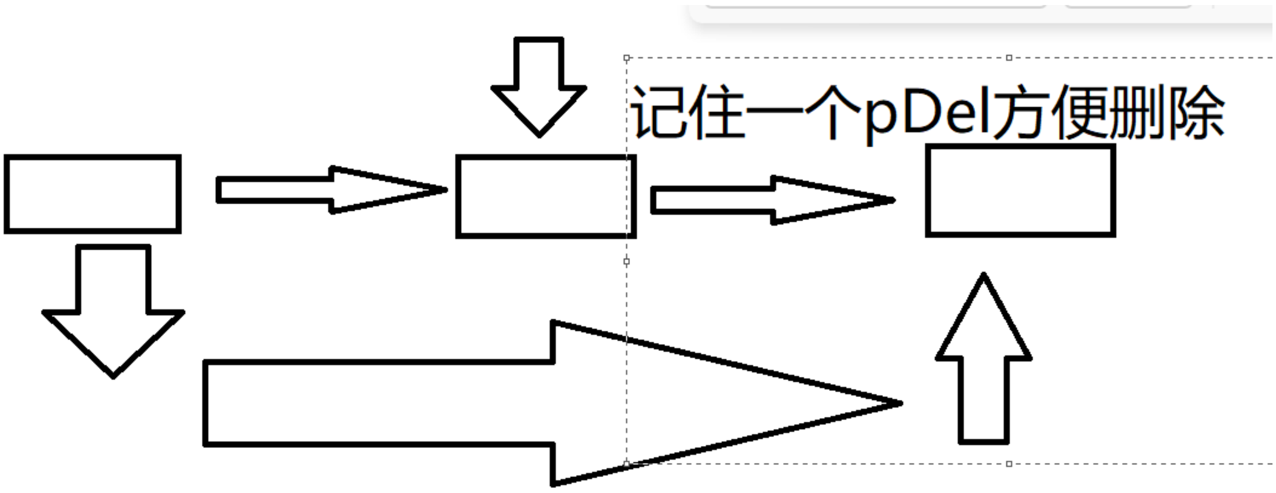
我们删除就不整什么尾删法了，直接上任意删除

```c
//-------------------------------------------Deletion_Functions-------------
// there are two basic type of functions in the sections
// 1. Erase a targeted element and erase it from the ClassicDataList
// fun1:eraseAElementfromDataList
//
// input : the list and the erase position
// output: the statues of normal ,can be used in check or just ignore!
//
// ------------------------------------------------------------------------
```

```
//
// 2.Erase some target elements from the ClassicDataList
// fun2:eraseAbunchData
//
// input: the list and the start position as well as the last position, that means offer a
paired pos
// output: the statues of normal ,can be used in check or just ignore!
//-----------------------------------------Deletion_Functions---------------
```

删除一个节点

对于删除一个元素，我们就逆向过程一下：



先指定pDel是查找指针的下一个：然后，记作 pDel 之后呢，再：删除与指针重连。

函数的原型显然是显而易见的：

```
ClassicLinkListFunctionStatues eraseAElementfromDataList(
    Classic_DataList*                                        list,
    size_t                                                   pos
)
```

做好数据检查：

```
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (pos<0 || pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
```

开始区分，如果是删除头节点：

```
    if (pos == 0)
    {
        Classic_DataNode* pNewHead = list->Head->p_next_one;
        Classic_DataNode* pDel = list->Head;
        list->Head = pNewHead;
        free(pDel->data);
        list->current_size--;
        return ClassicLinkList_NORMAL;
    }
```

之后的类似：

```
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < pos-1; i++)
    {
        pCur = pCur->p_next_one;
    }
    Classic_DataNode* pDel = pCur->p_next_one;
    pCur->p_next_one = pDel->p_next_one;
    free(pDel->data);
    free(pDel);
    list->current_size--;
    return ClassicLinkList_NORMAL;
```

```
// About ClassicDataList
// delete a  data in the datalist by offering a position
//used like this: eraseAElementfromDataList(list, erasing_pos)
//
ClassicLinkListFunctionStatues eraseAElementfromDataList(
    Classic_DataList*                                       list,
    size_t                                                  pos
)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (pos<0 || pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (pos == 0)
    {
        Classic_DataNode* pNewHead = list->Head->p_next_one;
        Classic_DataNode* pDel = list->Head;
        list->Head = pNewHead;
        free(pDel->data);
        list->current_size--;
        return ClassicLinkList_NORMAL;
    }
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < pos-1; i++)
    {
        pCur = pCur->p_next_one;
    }
    Classic_DataNode* pDel = pCur->p_next_one;
    pCur->p_next_one = pDel->p_next_one;
    free(pDel->data);
    free(pDel);
    list->current_size--;
    return ClassicLinkList_NORMAL;
}
```

删除多个节点

函数的原型显而易见了，不多赘述：

```
ClassicLinkListFunctionStatues eraseAbunchData_inDataList(
    Classic_DataList*                                            list,
    size_t                                                      front_pos,
    size_t                                                      final_pos
)
```

数据检查是少不了的：

```
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (front_pos<0 || front_pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (final_pos<0 || final_pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (front_pos > final_pos)
    {
        SHOW_WARNING_ClassicLinkList_SWAPPED_POS;
        SHOW_WHETHER_ACCCEPTED("y/n: y for yes , n for n");
        int choice;
        scanf("%c", &choice);
        if (choice == 'y')
        {
            int temp = final_pos;
            final_pos = front_pos;
            front_pos = temp;
        }
        else
        {
            SHOW_ERROR_ClassicLinkList_Invalid_Input;
            exit(ClassicLinkList_Invalid_Input);
        }
    }
```

仍然是区分是不是包含了头节点的问题：

```
    Bool flag =False;
    if (front_pos == 0)
    {
        flag = True;
        front_pos++;
    }
    //开始移动位置
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < front_pos-1; i++)
    {
        pCur = pCur->p_next_one;
    }
    int cur_place = front_pos-1;
    while (cur_place <= final_pos-1) {
        Classic_DataNode* pDel = pCur->p_next_one;
        pCur->p_next_one = pDel->p_next_one;
        free(pDel);
        cur_place++;
    }
    list->current_size -= final_pos - front_pos +1;

    //单独处理头节点的问题:
if (flag == True) {
    Classic_DataNode* pNewHead = pCur->p_next_one;
    Classic_DataNode* pDel = list->Head;
    list->Head = pNewHead;
    free(pDel->data);
    list->current_size--;
}
    return ClassicLinkList_NORMAL;
```

删除函数API一览

```
//------------------------------------------Deletion_Functions-------------
// there are two basic type of functions in the sections
// 1. Erase a targeted element and erase it from the ClassicDataList
// fun1:eraseAElementfromDataList
//
// input : the list and the erase position
// output: the statues of normal ,can be used in check or just ignore!
//
// ---------------------------------------------------------------------------
//
// 2.Erase some target elements from the ClassicDataList
// fun2:eraseAbunchData
//
// input: the list and the start position as well as the last position, that means offer a
paired pos
// output: the statues of normal ,can be used in check or just ignore!
//------------------------------------------Deletion_Functions----------------
```

```c
// About ClassicDataList
// delete a  data in the datalist by offering a position
//used like this: eraseAElementfromDataList(list, erasing_pos)
//
ClassicLinkListFunctionStatues eraseAElementfromDataList(
    Classic_DataList*                                         list,
    size_t                                                    pos
)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (pos<0 || pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (pos == 0)
    {
        Classic_DataNode* pNewHead = list->Head->p_next_one;
        Classic_DataNode* pDel = list->Head;
        list->Head = pNewHead;
        free(pDel->data);
        list->current_size--;
        return ClassicLinkList_NORMAL;
    }
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < pos-1; i++)
    {
        pCur = pCur->p_next_one;
    }
    Classic_DataNode* pDel = pCur->p_next_one;
    pCur->p_next_one = pDel->p_next_one;
    free(pDel->data);
    free(pDel);
    list->current_size--;
    return ClassicLinkList_NORMAL;
}


// About ClassicDataList
// delete a bunch of data in the datalist
// used like this eraseAbunchData( list,  front_pos,  final_pos)
//
ClassicLinkListFunctionStatues eraseAbunchData_inDataList(
    Classic_DataList*                                             list,
    size_t                                                        front_pos,
    size_t                                                        final_pos
)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (front_pos<0 || front_pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (final_pos<0 || final_pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (front_pos > final_pos)
    {
        SHOW_WARNING_ClassicLinkList_SWAPPED_POS;
        SHOW_WHETHER_ACCCEPTED("y/n: y for yes , n for n");
        int choice;
        scanf("%c", &choice);
        if (choice == 'y')
        {
            int temp = final_pos;
            final_pos = front_pos;
            front_pos = temp;
        }
        else
        {
            SHOW_ERROR_ClassicLinkList_Invalid_Input;
            exit(ClassicLinkList_Invalid_Input);
        }
    }
    Bool flag =False;
    if (front_pos == 0)
    {
        flag = True;
        front_pos++;
    }
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < front_pos-1; i++)
    {
        pCur = pCur->p_next_one;
    }
    int cur_place = front_pos-1;
    while (cur_place <= final_pos-1) {
        Classic_DataNode* pDel = pCur->p_next_one;
        pCur->p_next_one = pDel->p_next_one;
        free(pDel);
        cur_place++;
    }
    list->current_size -= final_pos - front_pos +1;
    if (flag == True) {
        Classic_DataNode* pNewHead = pCur->p_next_one;
        Classic_DataNode* pDel = list->Head;
        list->Head = pNewHead;
        free(pDel->data);
        front_pos++;
        list->current_size--;
    }
    return ClassicLinkList_NORMAL;
}
```

查询位置类函数

这个函数模块比较大了，我们首先设计一个是否存在的查询，其次还要设计返回位置的函数，甚至是返回多个位置的函数：

```
//----------------------------------------check_if_element_existed_functions-----
// there are four functions in this sections
//
// 1.check if the classiclinklist is empty
// fun1: isEmptyClassicLinkList
//
// input :the checked list
// output: My defined bool(Sadly the org C doesn't define that)
//
// ------------------------------------------------------------------------------
//
// 2. check if the target elements is exsited in the datalist
// fun2: checkIsLocateinLinkList
//
// input : the checked list ,the searched data and the Compared functions
// output: the locations that we first found it
//
// -------------------------------------------------------------------------------
//
// 3. return out the first locations that we found of the target elements
// func3:checkIsLocateinLinkList
//
// input : the checked list ,the searched data and the Compared functions
// output: My defined bool(Sadly the org C doesn't define that)
//
// --------------------------------------------------------------------------------
//
// Warning : this functions required the PSDA abstractions is opened so you can use
Position_Stored_Dynamic_ArrayFordyarr*
// to get the position
//
// 4. return out a bunch of data that was contained in Position_Stored_Dynamic_ArrayFordyarr
// fun4:returnAbunchLocationsinLinkList
//
// input : the checked list , the Position_Stored_Dynamic_ArrayFordyarr*(required init first)
// output: the Position_Stored_Dynamic_ArrayFordyarr that stored the positions!
//
// ------------------------------------------------------------------
```

链表是否为空函数

很简答，直接略

```
// About ClassicLinkList
// check if the LinkList is empty
//
Bool isEmptyClassicLinkList(Classic_DataList* list)
{
    if (list->current_size == 0)
    {
        return True;
    }
    return False;
}
```

对于元素是否存在，我看，我们的用户要定义什么是存在，就是说，提供一个函数可以描述需求：

使用 LocateFun 函数就好了

```
LocateFunc user_func;
```

```
// About ClassicLinkList
// check if the target elements is exsited in the datalist
//
Bool checkIsLocateinLinkList(
    Classic_DataList*                                         list,
    void*                                                     data,
    LocateFunc                                           user_func
)
```

检查输入的合法性：

```
if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (data == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (user_func == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
```

走一遍，比较一遍就好了：

```
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < list->current_size; i++)
    {
        if (user_func(data, pCur->data)) {
            return True;
        }
    }
    return False;
```

返回具体位置？把存在改成返回位置就好了！

```
// About ClassicLinkList
// return out the first locations that we found of the target elements
// can use in this way: size_t pos = returnOutDatabyposinLinkList( list, data, user_func)
size_t returnOutDatabyposinLinkList(
    Classic_DataList*                                       list,
    void*                                                   data,
    LocateFunc                                              user_func
)
{
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < list->current_size; i++)
    {
        if (user_func(data, pCur->data)) {
            return i;
        }
        pCur = pCur->p_next_one;
    }
    return Unfind_ClassicLinkList;
}
```

```
    下面的多重返回：需要打开动态数组存储
```

```
// About ClassicLinkList
// return out a bunch of data that was contained in Position_Stored_Dynamic_ArrayFordyarr
// can use in this way: Position_Stored_Dynamic_ArrayFordyarr* somePos =
// returnAbunchLocationsinLinkList( list,  posArr,  data,  user_func)
//
Position_Stored_Dynamic_ArrayFordyarr* returnAbunchLocationsinLinkList(
    Classic_DataList*                                   list,
    Position_Stored_Dynamic_ArrayFordyarr*                posArr,
    void*                                              data,
    LocateFunc                                          user_func
)
```

```
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (data == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (posArr == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* p_Currency = list->Head;
    int possible_count = 0;
    for (int i = 0; i < list->current_size; i++)
    {
        if ((*user_func)(data, p_Currency->data))
        {
            //开始重新开辟下内存
            int* pro_usable_stored_space = (int*)realloc(posArr->posSpace, sizeof(int) *
(possible_count + 1));
            if (!pro_usable_stored_space)
            {
                SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
                exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
            }
            posArr->posSpace = pro_usable_stored_space;
            //向第I个位置存储
            posArr->posSpace[possible_count] = i;
            possible_count++;
            posArr->pos_size = possible_count;
        }
        p_Currency = p_Currency->p_next_one;
    }
    if (posArr->pos_size)
        return posArr;
    return Unfind_ClassicLinkList;
}
```

查询函数一览

```
//----------------------------------------check_if_element_existed_functions-----
// there are four functions in this sections
//
// 1.check if the classiclinklist is empty
// fun1: isEmptyClassicLinkList
//
// input :the checked list
// output: My defined bool(Sadly the org C doesn't define that)
//
// ------------------------------------------------------------------------------
//
// 2. check if the target elements is exsited in the datalist
// fun2: checkIsLocateinLinkList
//
// input : the checked list ,the searched data and the Compared functions
// output: the locations that we first found it
```

```
//
// --------------------------------------------------------------------------------
//
// 3. return out the first locations that we found of the target elements
// func3:checkIsLocateinLinkList
//
// input : the checked list ,the searched data and the Compared functions
// output: My defined bool(Sadly the org C doesn't define that)
//
// ---------------------------------------------------------------------------------
//
// Warning : this functions required the PSDA abstractions is opened so you can use
Position_Stored_Dynamic_ArrayFordyarr*
// to get the position
//
// 4. return out a bunch of data that was contained in Position_Stored_Dynamic_ArrayFordyarr
// fun4:returnAbunchLocationsinLinkList
//
// input : the checked list , the Position_Stored_Dynamic_ArrayFordyarr*(required init first)
// output: the Position_Stored_Dynamic_ArrayFordyarr that stored the positions!
//
// ----------------------------------------------------------------------------------



// About ClassicLinkList
// check if the LinkList is empty
//
Bool isEmptyClassicLinkList(Classic_DataList* list)
{
    if (list->current_size == 0)
    {
        return True;
    }
    return False;
}

// About ClassicLinkList
// check if the target elements is exsited in the datalist
//
Bool checkIsLocateinLinkList(
    Classic_DataList*                                                list,
    void*                                                            data,
    LocateFunc                                                       user_func
)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (data == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (user_func == NULL)
    {
```

```c
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < list->current_size; i++)
    {
        if (user_func(data, pCur->data)) {
            return True;
        }
    }
    return False;
}

// About ClassicLinkList
// return out the first locations that we found of the target elements
// can use in this way: size_t pos = returnOutDatabyposinLinkList( list, data, user_func)
size_t returnOutDatabyposinLinkList(
    Classic_DataList*                                        list,
    void*                                                    data,
    LocateFunc                                               user_func
)
{
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < list->current_size; i++)
    {
        if (user_func(data, pCur->data)) {
            return i;
        }
        pCur = pCur->p_next_one;
    }
    return Unfind_ClassicLinkList;
}


// Make sure the OPENUPPOSARR == 1 if wanted to use it

#if OPENUPPOSARR

// About ClassicLinkList
// return out a bunch of data that was contained in Position_Stored_Dynamic_ArrayFordyarr
// can use in this way: Position_Stored_Dynamic_ArrayFordyarr* somePos =
// returnAbunchLocationsinLinkList( list,  posArr,  data,  user_func)
//
Position_Stored_Dynamic_ArrayFordyarr* returnAbunchLocationsinLinkList(
    Classic_DataList*                                        list,
    Position_Stored_Dynamic_ArrayFordyarr*                   posArr,
    void*                                                    data,
    LocateFunc                                               user_func
)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (data == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (posArr == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* p_Currency = list->Head;
    int possible_count = 0;
    for (int i = 0; i < list->current_size; i++)
    {
        if ((*user_func)(data, p_Currency->data))
        {
            int* pro_usable_stored_space = (int*)realloc(posArr->posSpace, sizeof(int) *
(possible_count + 1));
            if (!pro_usable_stored_space)
            {
                SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
                exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
            }
            posArr->posSpace = pro_usable_stored_space;
            posArr->posSpace[possible_count] = i;
            possible_count++;
            posArr->pos_size = possible_count;
        }
        p_Currency = p_Currency->p_next_one;
    }
    if (posArr->pos_size)
        return posArr;
    return Unfind_ClassicLinkList;
}

#endif
```

一些作用函数

请看一览：

```
//--------------------------------------Do_Something_to_the_datalist_functions----------
// there are seven functions in this sections
//
// 1. Classic Print functions
// func1: Print_All_Data_LinkList
// to print all the data in the user_defined linklist
//
// input : list and the user print , and the decision whether enjoyed a better print
// output: the statues of normal ,can be used in check or just ignore!
//
// ------------------------------------------------------------------------------------
//
// 2. Do some change to the specific data
// to make some change directly to an element
//
// input : the list ,the targeted positions and the way how you want to operate the data itself
// output: the statues of normal ,can be used in check or just ignore!
//
// ------------------------------------------------------------------------------------
//
// 3. Do some change to all the data in the classiclinklist
// to make some change directly to the whole elements
//
// input : the list ,the way how you want to operate the data itself
// output: the statues of normal ,can be used in check or just ignore!
//
// -----------------------------------------------------------------------------------
//
// 4. Do specific change to an interval pos in the classicLickList
// to make some change directly to the interval elements
//
// input : the list ,the front_pos and the final_pos
// output: the statues of normal ,can be used in check or just ignore!
//
// -----------------------------------------------------------------------------------
//
// 5. reverse ClassicLinkList
// to reverse up the classic LinkList
//
// input : the ready_reversed data
// output: the statues of normal ,can be used in check or just ignore!
//
// -----------------------------------------------------------------------------------
//
// 6. combine two linkList into one
// to combine the two classic linklist into one!
//
// input : the two linklist
// output: the merged linklist Pointer
//
// -----------------------------------------------------------------------------------
//
// 7. sort the elements in the classicLinkList under the user's constructions
// to sort the ClassicLinkList
//
// input : the classicLinklist
// output: the statues of normal ,can be used in check or just ignore!
//
// --------------------------------------------------------------------
```

打印函数

还是遍历链表，由于跟查询类似就不再多费口舌了：

```
// About ClassicLinkList
// to print the data
// use in this way: Print_All_Data_LinkList( list, user_print,  whether_better_print)
//
ClassicLinkListFunctionStatues Print_All_Data_LinkList(
    Classic_DataList*                          list,
    MyPrint                                     user_print,
    BetterPrintSwitch_for_ClassicLinkList        whether_better_print
)
{

    if (!list)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }

    if (!user_print)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }

    if (whether_better_print == ClassicLinkList_OPEN) {
        printf("\nStart printing:\n");
    }


    Classic_DataNode* p_Currency = list->Head;

    for (int i = 0; i < list->current_size; i++)
    {
        user_print(p_Currency->data);
        p_Currency = p_Currency->p_next_one;
    }


    if (whether_better_print == ClassicLinkList_OPEN) {
        printf("\nFinish printing:\n");
    }

    return ClassicLinkList_NORMAL;
}
```

三大作用函数

个体作用函数

跟查询单个类似：

```
// About ClassicLinkList
// to make some change to the specific data
// use in this way:doChangetoSpecificElementinLinkList( list,  pos , user_func)
//
ClassicLinkListFunctionStatues doChangetoSpecificElementinLinkList(
    Classic_DataList*                                list,
    size_t                                           pos ,
    Do_Specific_Change                               user_func
)
{
    // 数据合法性
    if (list == NULL&&user_func==NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (pos<0 || pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    // 开始寻找对应的
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < pos; i++)
    {
        pCur = pCur->p_next_one;
    }
    user_func(pCur->data);
    return ClassicLinkList_NORMAL;
}
```

区间作用函数

不难想的：提供作用方式 + 区间长度+区间位置

```
// About ClassicLinkList
// to make some change to the interval data
// use in this way:doChangetoTargetIntervalDatainLinkList(list , pos1,pos2, user_func)
//
ClassicLinkListFunctionStatues doChangetoTargetIntervalDatainLinkList(
    Classic_DataList*                                       list,
    size_t                                              front_pos,
    size_t                                              final_pos,
    Do_Specific_Change                                  user_func
)
```

还是类似的：

```
    // 数据检查
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (front_pos<0 || front_pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (final_pos<0 || final_pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (front_pos > final_pos)
    {
        SHOW_WARNING_ClassicLinkList_SWAPPED_POS;
        SHOW_WHETHER_ACCCEPTED("y/n: y for yes , n for n");
        int choice;
        scanf("%c", &choice);
        if (choice == 'y')
        {
            int temp = final_pos;
            final_pos = front_pos;
            front_pos = temp;
        }
        else
        {
            SHOW_ERROR_ClassicLinkList_Invalid_Input;
            exit(ClassicLinkList_Invalid_Input);
        }
    }

    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < front_pos; i++) {
        pCur = pCur->p_next_one;
    }
    for (int i = front_pos; i < final_pos+1; i++)
    {
        user_func(pCur->data);
        pCur = pCur->p_next_one;
    }
    return ClassicLinkList_NORMAL;
```

全体作用

不细说，类似的：

```
// About ClassicLinkList
// to make some change to the whole elements in the data
// use in this way:doChangetoAllDatainLinkList( list,  user_func)
//
ClassicLinkListFunctionStatues doChangetoAllDatainLinkList(
    Classic_DataList*                                   list,
    Do_Specific_Change                                  user_func
)
{
    if (list == NULL && user_func == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < list->current_size; i++)
    {
        user_func(pCur->data);
        pCur=pCur->p_next_one;
    }
    return ClassicLinkList_NORMAL;
}
```

三大函数API一览

```
// About ClassicLinkList
// to make some change to the specific data
// use in this way:doChangetoSpecificElementinLinkList( list,  pos , user_func)
//
ClassicLinkListFunctionStatues doChangetoSpecificElementinLinkList(
    Classic_DataList*                                list,
    size_t                                           pos ,
    Do_Specific_Change                               user_func
)
{
    if (list == NULL&&user_func==NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (pos<0 || pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < pos; i++)
    {
        pCur = pCur->p_next_one;
    }
    user_func(pCur->data);
    return ClassicLinkList_NORMAL;
}

// About ClassicLinkList
// to make some change to the interval data
// use in this way:doChangetoTargetIntervalDatainLinkList(list , pos1,pos2, user_func)
//
ClassicLinkListFunctionStatues doChangetoTargetIntervalDatainLinkList(
    Classic_DataList*                                        list,
    size_t                                                   front_pos,
    size_t                                                   final_pos,
    Do_Specific_Change                                       user_func
)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (front_pos<0 || front_pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (final_pos<0 || final_pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
```

```c
    if (front_pos > final_pos)
    {
        SHOW_WARNING_ClassicLinkList_SWAPPED_POS;
        SHOW_WHETHER_ACCCEPTED("y/n: y for yes , n for n");
        int choice;
        scanf("%c", &choice);
        if (choice == 'y')
        {
            int temp = final_pos;
            final_pos = front_pos;
            front_pos = temp;
        }
        else
        {
            SHOW_ERROR_ClassicLinkList_Invalid_Input;
            exit(ClassicLinkList_Invalid_Input);
        }
    }

    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < front_pos; i++) {
        pCur = pCur->p_next_one;
    }
    for (int i = front_pos; i < final_pos+1; i++)
    {
        user_func(pCur->data);
        pCur = pCur->p_next_one;
    }
    return ClassicLinkList_NORMAL;

}

// About ClassicLinkList
// to make some change to the whole elements in the data
// use in this way:doChangetoAllDatainLinkList( list,  user_func)
//
ClassicLinkListFunctionStatues doChangetoAllDatainLinkList(
    Classic_DataList*                                   list,
    Do_Specific_Change                                  user_func
)
{
    if (list == NULL && user_func == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < list->current_size; i++)
    {
        user_func(pCur->data);
        pCur=pCur->p_next_one;
    }
    return ClassicLinkList_NORMAL;
}
```

翻转链表函数

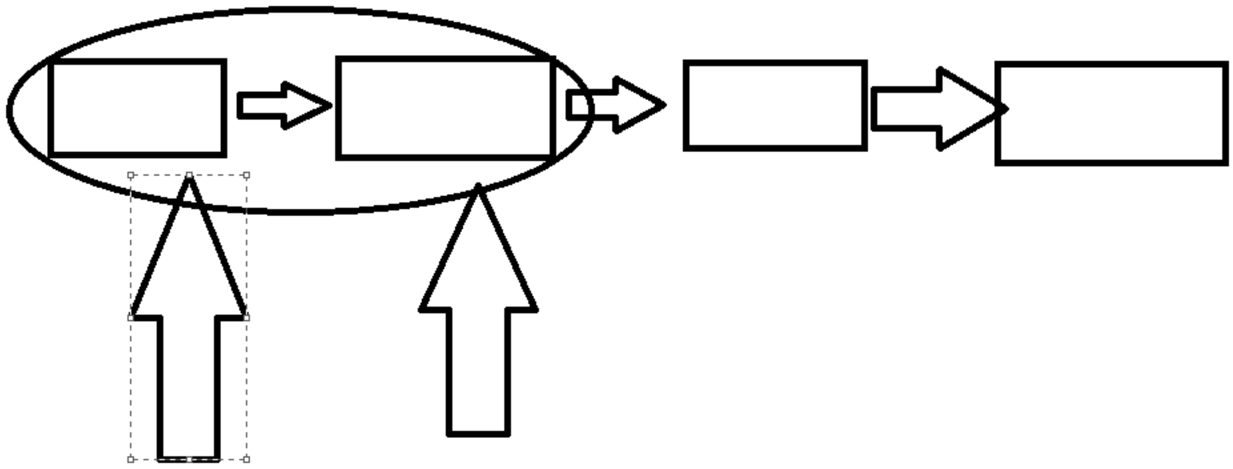
看一个反转链表的函数

为了加快效率，我们分类讨论一下：

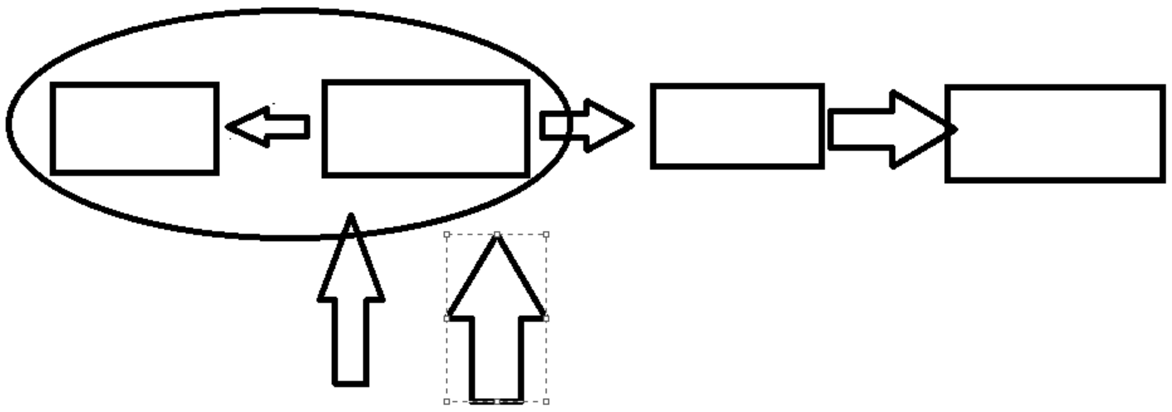
一个没必要反转，两个直接互换就好，问题在于多个的：

```
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (list->current_size == 1)
    {
        printf("No need in reversing!,try a longer one lol");
        return ClassicLinkList_NORMAL;
    }
    if (list->current_size == 2)
    {
        Classic_DataNode* headFisrt = list->Head;
        Classic_DataNode* pFin = headFisrt->p_next_one;
        pFin->p_next_one = headFisrt;
        headFisrt->p_next_one = NULL;
        list->Head = pFin;
        return ClassicLinkList_NORMAL;
    }
```

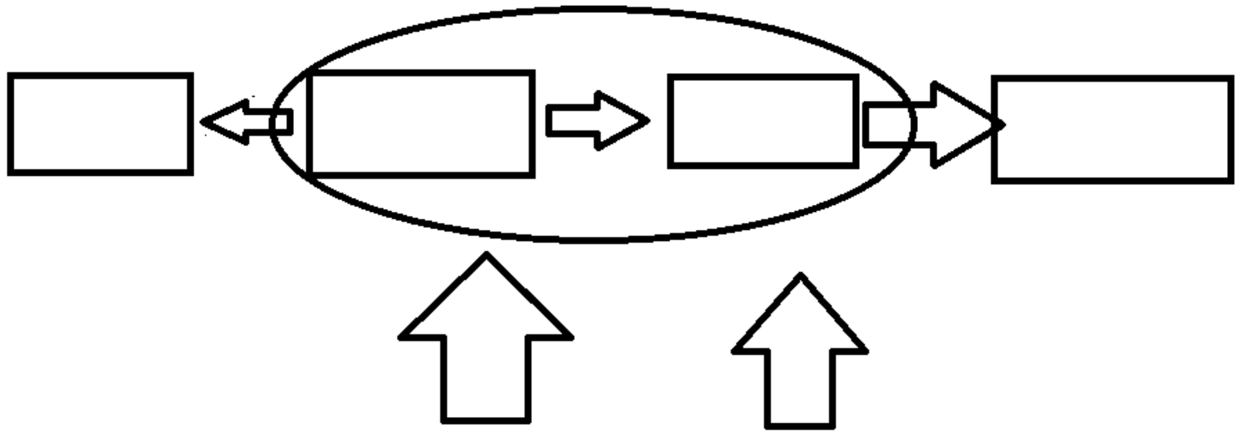
对于一个中间片段：



我们需要首先将指针反转，然后，后行指针先被先行指针赋值：



先行指针后移：成为子问题：



代码上就是这样的：

```
// About ClassicLinkList
// to reverse up the classiclinklist
// use in this way:ReverseLinkList(list)
//
ClassicLinkListFunctionStatues ReverseLinkList(Classic_DataList* list)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (list->current_size == 1)
    {
        printf("No need in reversing!,try a longer one lol");
        return ClassicLinkList_NORMAL;
    }
    if (list->current_size == 2)
    {
        Classic_DataNode* headFisrt = list->Head;
        Classic_DataNode* pFin = headFisrt->p_next_one;
        pFin->p_next_one = headFisrt;
        headFisrt->p_next_one = NULL;
        list->Head = pFin;
        return ClassicLinkList_NORMAL;
    }
    Classic_DataNode* pFinal = list->Head->p_next_one;
    Classic_DataNode* pPrior = list->Head;
    while (pFinal!= NULL)
    {
        Classic_DataNode* tempFin = pFinal->p_next_one;
        pFinal->p_next_one = pPrior;
        pPrior = pFinal;
        pFinal = tempFin;
    }
    list->Head = pPrior;
}
```

链表的合并于衔接函数

对于这样，仍然是先检查合法性+指向末尾：

况且，如果用户不想拷贝数据，直接传参

```
if (list_be_added == NULL && list_adder == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* pCur=list_be_added->Head;
    for (int i = 0; i < list_be_added->current_size-1; i++)
    {
        pCur = pCur->p_next_one;
    }
    //Do not copy , instead just merge!
    if (whether_copied==ClassicLinkList_NO_COPY) {
        pCur->p_next_one = list_adder->Head;
        list_be_added->current_size += list_adder->current_size;
        return list_be_added;
    }
```

拷贝的话，就是 copy + 衔接的问题：

```
Classic_DataNode* pAdder = list_adder->Head;
    for (int i = 0; i < list_adder->current_size; i++)
    {
        //prepare the space
        Classic_DataNode* pCopy = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        void* dataCopy = malloc(sizeof(pAdder->data));
        if (pCopy == NULL && dataCopy == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        //Using to copy data and initialize the copied node
        memcpy(dataCopy, pAdder->data, sizeof(pAdder->data));
        pCopy->data = dataCopy;
        pCopy->p_next_one = NULL;
        //then push back into the linklist
        pCur->p_next_one = pCopy;
        //Moving the copy pointer
        pAdder = pAdder->p_next_one;
        pCur = pCur->p_next_one;
    }
    list_be_added->current_size += list_adder->current_size;
    return list_be_added;
```

看看：

```
// About ClassicLinkList
// to merge two linklist into the one
// use in this way:mergeLinkList(list_be_added, list_adder, whether_copied)
//
Classic_DataList* mergeLinkList(
    Classic_DataList*                                    list_be_added,
    Classic_DataList*                                    list_adder,
    size_t                                               whether_copied
)
{
    if (list_be_added == NULL && list_adder == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* pCur=list_be_added->Head;
    for (int i = 0; i < list_be_added->current_size-1; i++)
    {
        pCur = pCur->p_next_one;
    }
    //Do not copy , instead just merge!
    if (whether_copied==ClassicLinkList_NO_COPY) {
        pCur->p_next_one = list_adder->Head;
        list_be_added->current_size += list_adder->current_size;
        return list_be_added;
    }
    Classic_DataNode* pAdder = list_adder->Head;
    for (int i = 0; i < list_adder->current_size; i++)
    {
        //prepare the space
        Classic_DataNode* pCopy = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        void* dataCopy = malloc(sizeof(pAdder->data));
        if (pCopy == NULL && dataCopy == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        //Using to copy data and initialize the copied node
        memcpy(dataCopy, pAdder->data, sizeof(pAdder->data));
        pCopy->data = dataCopy;
        pCopy->p_next_one = NULL;
        //then push back into the linklist
        pCur->p_next_one = pCopy;
        //Moving the copy pointer
        pAdder = pAdder->p_next_one;
        pCur = pCur->p_next_one;
    }
    list_be_added->current_size += list_adder->current_size;
    return list_be_added;
}
```

排序整理函数

使用冒泡排序排序我们的链表，两两的比较，本质上就是 pCur 跟 pCur->next_one 比较。。。那就很简单了：

```
// About ClassicLinkList
// sort the classicLinklist in the bobblesort
// use in this way:sortClassicLinkListinBubbleSort(Classic_DataList* list, CompareFunc
user_func)
//
ClassicLinkListFunctionStatues sortClassicLinkListinBubbleSort(
    Classic_DataList*                                                    list,
    CompareFunc                                                          user_func
)
{
    //检查数据是否合法
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (user_func == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    for (int i = 0; i < list->current_size-1; i++)
    {
        Classic_DataNode* pCur = list->Head;
        for(int j=0;j<list->current_size-i-1;j++)
        {
            //this two for loops are just standard BobbleSort
            if (user_func(pCur->data, pCur->p_next_one->data))
            {
                void* tempdata = malloc(sizeof(pCur->data));
                memcpy(tempdata, pCur->data, sizeof(pCur->data));
                int tempdatasize = sizeof(pCur->data);
                //Before Swap,make sure that the dataContainer is huge enough to contain a new
space
                if (sizeof(pCur->p_next_one->data) > sizeof(pCur->data))
                {
                    void* pTempSwap = realloc(pCur->data, sizeof(pCur->p_next_one->data));
                    if (pTempSwap == NULL)
                    {
                        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
                        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
                    }
                    pCur->data = pTempSwap;
                }
                //Copy and swap
                memcpy(pCur->data, pCur->p_next_one->data, sizeof(pCur->p_next_one->data));
                memcpy(pCur->p_next_one->data, tempdata, tempdatasize);
            }
            //Moving pointer to the next one
            pCur = pCur->p_next_one;
        }
    }
    return ClassicLinkList_NORMAL;
}
```

作用函数一览：

```
//---------------------------------------Do_Something_to_the_datalist_functions----------
// there are seven functions in this sections
//
// 1. Classic Print functions
// func1: Print_All_Data_LinkList
// to print all the data in the user_defined linklist
//
// input : list and the user print , and the decision whether enjoyed a better print
// output: the statues of normal ,can be used in check or just ignore!
//
// ------------------------------------------------------------------------------------
//
// 2. Do some change to the specific data
// to make some change directly to an element
//
// input : the list ,the targeted positions and the way how you want to operate the data itself
// output: the statues of normal ,can be used in check or just ignore!
//
// ------------------------------------------------------------------------------------
//
// 3. Do some change to all the data in the classiclinklist
// to make some change directly to the whole elements
//
// input : the list ,the way how you want to operate the data itself
// output: the statues of normal ,can be used in check or just ignore!
//
// -----------------------------------------------------------------------------------
//
// 4. Do specific change to an interval pos in the classicLickList
// to make some change directly to the interval elements
//
// input : the list ,the front_pos and the final_pos
// output: the statues of normal ,can be used in check or just ignore!
//
// -----------------------------------------------------------------------------------
//
// 5. reverse ClassicLinkList
// to reverse up the classic LinkList
//
// input : the ready_reversed data
```

```c
// output: the statues of normal ,can be used in check or just ignore!
//
// ----------------------------------------------------------------------------------
//
// 6. combine two linkList into one
// to combine the two classic linklist into one!
//
// input : the two linklist
// output: the merged linklist Pointer
//
// ----------------------------------------------------------------------------------
//
// 7. sort the elements in the classicLinkList under the user's constructions
// to sort the ClassicLinkList
//
// input : the classicLinklist
// output: the statues of normal ,can be used in check or just ignore!
//
// --------------------------------------------------------------------------------


// About ClassicLinkList
// to print the data
// use in this way: Print_All_Data_LinkList( list, user_print,  whether_better_print)
//
ClassicLinkListFunctionStatues Print_All_Data_LinkList(
    Classic_DataList*                                   list,
    MyPrint                                             user_print,
    BetterPrintSwitch_for_ClassicLinkList               whether_better_print
)
{

    if (!list)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }

    if (!user_print)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }

    if (whether_better_print == ClassicLinkList_OPEN) {
        printf("\nStart printing:\n");
    }


    Classic_DataNode* p_Currency = list->Head;

    for (int i = 0; i < list->current_size; i++)
    {
        user_print(p_Currency->data);
        p_Currency = p_Currency->p_next_one;
    }
    if (whether_better_print == ClassicLinkList_OPEN) {
        printf("\nFinish printing:\n");
    }

    return ClassicLinkList_NORMAL;
}

// About ClassicLinkList
// to make some change to the specific data
// use in this way:doChangetoSpecificElementinLinkList( list,  pos , user_func)
//
ClassicLinkListFunctionStatues doChangetoSpecificElementinLinkList(
    Classic_DataList*                                   list,
    size_t                                              pos ,
    Do_Specific_Change                                  user_func
)
{
    if (list == NULL&&user_func==NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (pos<0 || pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < pos; i++)
    {
        pCur = pCur->p_next_one;
    }
    user_func(pCur->data);
    return ClassicLinkList_NORMAL;
}

// About ClassicLinkList
// to make some change to the interval data
// use in this way:doChangetoTargetIntervalDatainLinkList(list , pos1,pos2, user_func)
//
ClassicLinkListFunctionStatues doChangetoTargetIntervalDatainLinkList(
    Classic_DataList*                                           list,
    size_t                                                      front_pos,
    size_t                                                      final_pos,
    Do_Specific_Change                                          user_func
)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (front_pos<0 || front_pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (final_pos<0 || final_pos>list->current_size)
    {
        SHOW_ERROR_ClassicLinkList_Invalid_Input;
        exit(ClassicLinkList_Invalid_Input);
    }
    if (front_pos > final_pos)
    {
        SHOW_WARNING_ClassicLinkList_SWAPPED_POS;
        SHOW_WHETHER_ACCCEPTED("y/n: y for yes , n for n");
        int choice;
        scanf("%c", &choice);
        if (choice == 'y')
        {
            int temp = final_pos;
            final_pos = front_pos;
            front_pos = temp;
        }
        else
        {
            SHOW_ERROR_ClassicLinkList_Invalid_Input;
            exit(ClassicLinkList_Invalid_Input);
        }
    }

    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < front_pos; i++) {
        pCur = pCur->p_next_one;
    }
    for (int i = front_pos; i < final_pos+1; i++)
    {
        user_func(pCur->data);
        pCur = pCur->p_next_one;
    }
    return ClassicLinkList_NORMAL;

}

// About ClassicLinkList
// to make some change to the whole elements in the data
// use in this way:doChangetoAllDatainLinkList( list,  user_func)
//
ClassicLinkListFunctionStatues doChangetoAllDatainLinkList(
    Classic_DataList*                                   list,
    Do_Specific_Change                                  user_func
)
{
    if (list == NULL && user_func == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* pCur = list->Head;
    for (int i = 0; i < list->current_size; i++)
    {
        user_func(pCur->data);
        pCur=pCur->p_next_one;
    }
    return ClassicLinkList_NORMAL;
}
```

```c
// About ClassicLinkList
// to reverse up the classiclinklist
// use in this way:ReverseLinkList(list)
//
ClassicLinkListFunctionStatues ReverseLinkList(Classic_DataList* list)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (list->current_size == 1)
    {
        printf("No need in reversing!,try a longer one lol");
        return ClassicLinkList_NORMAL;
    }
    if (list->current_size == 2)
    {
        Classic_DataNode* headFisrt = list->Head;
        Classic_DataNode* pFin = headFisrt->p_next_one;
        pFin->p_next_one = headFisrt;
        headFisrt->p_next_one = NULL;
        list->Head = pFin;
        return ClassicLinkList_NORMAL;
    }
    Classic_DataNode* pFinal = list->Head->p_next_one;
    Classic_DataNode* pPrior = list->Head;
    while (pFinal!= NULL)
    {
        Classic_DataNode* tempFin = pFinal->p_next_one;
        pFinal->p_next_one = pPrior;
        pPrior = pFinal;
        pFinal = tempFin;
    }
    list->Head = pPrior;
}

// About ClassicLinkList
// to merge two linklist into the one
// use in this way:mergeLinkList(list_be_added, list_adder, whether_copied)
//
Classic_DataList* mergeLinkList(
    Classic_DataList*                                   list_be_added,
    Classic_DataList*                                   list_adder,
    size_t                                              whether_copied
)
{
    if (list_be_added == NULL && list_adder == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    Classic_DataNode* pCur=list_be_added->Head;
    for (int i = 0; i < list_be_added->current_size-1; i++)
    {
        pCur = pCur->p_next_one;
    }
    //Do not copy , instead just merge!
    if (whether_copied==ClassicLinkList_NO_COPY) {
        pCur->p_next_one = list_adder->Head;
        list_be_added->current_size += list_adder->current_size;
        return list_be_added;
    }
    Classic_DataNode* pAdder = list_adder->Head;
    for (int i = 0; i < list_adder->current_size; i++)
    {
        //prepare the space
        Classic_DataNode* pCopy = (Classic_DataNode*)malloc(sizeof(Classic_DataNode));
        void* dataCopy = malloc(sizeof(pAdder->data));
        if (pCopy == NULL && dataCopy == NULL)
        {
            SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
            exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
        }
        //Using to copy data and initialize the copied node
        memcpy(dataCopy, pAdder->data, sizeof(pAdder->data));
        pCopy->data = dataCopy;
        pCopy->p_next_one = NULL;
        //then push back into the linklist
        pCur->p_next_one = pCopy;
        //Moving the copy pointer
        pAdder = pAdder->p_next_one;
        pCur = pCur->p_next_one;
    }
    list_be_added->current_size += list_adder->current_size;
    return list_be_added;
}

// About ClassicLinkList
// sort the classicLinklist in the bobblesort
// use in this way:sortClassicLinkListinBubbleSort(Classic_DataList* list, CompareFunc
user_func)
//
ClassicLinkListFunctionStatues sortClassicLinkListinBubbleSort(
    Classic_DataList*                                                          list,
    CompareFunc                                                                user_func
)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    if (user_func == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }
    for (int i = 0; i < list->current_size-1; i++)
    {
        Classic_DataNode* pCur = list->Head;
        for(int j=0;j<list->current_size-i-1;j++)
        {
            //this two for loops are just standard BobbleSort
            if (user_func(pCur->data, pCur->p_next_one->data))
            {
                void* tempdata = malloc(sizeof(pCur->data));
                memcpy(tempdata, pCur->data, sizeof(pCur->data));
                int tempdatasize = sizeof(pCur->data);
                //Before Swap,make sure that the dataContainer is huge enough to contain a new
space
                if (sizeof(pCur->p_next_one->data) > sizeof(pCur->data))
                {
                    void* pTempSwap = realloc(pCur->data, sizeof(pCur->p_next_one->data));
                    if (pTempSwap == NULL)
                    {
                        SHOW_ERROR_ClassicLinkList_ERROR_IN_MALLOCING_SPACE;
                        exit(ClassicLinkList_ERROR_IN_MALLOCING_SPACE);
                    }
                    pCur->data = pTempSwap;
                }
                //Copy and swap
                memcpy(pCur->data, pCur->p_next_one->data, sizeof(pCur->p_next_one->data));
                memcpy(pCur->p_next_one->data, tempdata, tempdatasize);
            }
            //Moving pointer to the next one
            pCur = pCur->p_next_one;
        }
    }
    return ClassicLinkList_NORMAL;
}
```

消除链表

可以看作一个区域删除的扩展

```
//---------------------------------------------------Basic_Erase------------------------------
//
// 1.Erase A LinkList
// used in just clear a linklist
// func1: clearAClassicLinkList
//
// input : just a classiclinklist
// output: the statues of normal ,can be used in check or just ignore!
//
// ---------------------------------------------------------------------------------------------

//About ClassicLinkList
// Erase a classicLinkList
// Use in this way clearAClassicLinkList( list)
//
ClassicLinkListFunctionStatues clearAClassicLinkList(Classic_DataList* list)
{
    if (list == NULL)
    {
        SHOW_ERROR_ClassicLinkList_NULL_INPUT;
        exit(ClassicLinkList_NULL_INPUT);
    }

    Classic_DataNode* pCur = list->Head->p_next_one;
    Classic_DataNode* pDel = list->Head;
    if (list->current_size == 1) {
        free(pDel);
    }
    for (int i = 0; i < list->current_size-1; i++)
    {
        free(pDel);
        pDel = pCur;
        pCur = pCur->p_next_one;
    }

    list->current_size = 0 ;
    return ClassicLinkList_NORMAL;
}
```

测试文档

可以把这几个头文件和源文件添加到工程测试:

```
#define _CRT_SECURE_NO_WARNINGS 1
#include"ClassicLinkList.h"

MyPrint print(int* e) {
    printf("%d ", *e);
}

Do_Specific_Change dochange(int* e) {
    *e = *e + 1;
}

LocateFunc loc(int* e1, int* e2) {
    return *e1 == *e2;
}

CompareFunc comp(int* e1, int* e2) {
    return *e1 < *e2;
}
int main()
{
    //do test initailization

    //default init
    Classic_DataList* defaultinit = Init_A_ClassicLinkList();

    //updata init

    int array[10] = { 1,2,3,4,5,6,7,8,9,10 };
    Classic_DataList* update_one = UpdateStaticArray2ClassicLinkList(array, Dyarr_SIGINT, 10);
    Print_All_Data_LinkList(update_one, print, ClassicLinkList_OPEN);

    // copy init

    Classic_DataList* copy_one = Init_A_ClassicLinkList_By_CopyAClassicLinkList(update_one);
    printf("\nThouth , this is the copy one:");
    Print_All_Data_LinkList(copy_one, print, ClassicLinkList_OPEN);
```

```c
// Push back
int addersig = 100;
printf("\nFirst push:");
Push_Back_Into_A_ClassicLinkList(defaultinit, &addersig, Dyarr_SIGINT);
Print_All_Data_LinkList(defaultinit, print, ClassicLinkList_OPEN);
printf("\nSecond push:");
Push_Back_Into_A_ClassicLinkList(defaultinit, &addersig, Dyarr_SIGINT);
Print_All_Data_LinkList(defaultinit, print, ClassicLinkList_OPEN);

// Push back a sum
int adderarr[5] = { 101,102,103,104,105 };
push_Back_BunchDataintoClassicLinkList(defaultinit, adderarr, Dyarr_SIGINT, 5);
Print_All_Data_LinkList(defaultinit, print, ClassicLinkList_OPEN);

//Insert start:
int insertionsig = -100;

//to the head:
Insert_into_AClassicLinkList(defaultinit, &insertionsig, Dyarr_SIGINT, 0);
Print_All_Data_LinkList(defaultinit, print, ClassicLinkList_OPEN);

//to other place
Insert_into_AClassicLinkList(defaultinit, &insertionsig, Dyarr_SIGINT, 1);
Print_All_Data_LinkList(defaultinit, print, ClassicLinkList_OPEN);

//for a bunch ...
//head
int insertarr[5] = { -1,-2,-3,-4,-5 };
Insert_A_Bunch_of_data_intoClassicLinkList(update_one, 0, insertarr, Dyarr_SIGINT, 5);
Print_All_Data_LinkList(update_one, print, ClassicLinkList_OPEN);

//other place
Insert_A_Bunch_of_data_intoClassicLinkList(update_one, 1, insertarr, Dyarr_SIGINT, 5);
Print_All_Data_LinkList(update_one, print, ClassicLinkList_OPEN);

//Erase
eraseAElementfromDataList(copy_one, 0);
Print_All_Data_LinkList(copy_one, print, ClassicLinkList_OPEN);

eraseAElementfromDataList(copy_one, 1);
Print_All_Data_LinkList(copy_one, print, ClassicLinkList_OPEN);

//Erase bunch
eraseAbunchData_inDataList(copy_one, 0, 2);
Print_All_Data_LinkList(copy_one, print, ClassicLinkList_OPEN);


//make change:
//sig
doChangetoSpecificElementinLinkList(copy_one, 0, dochange);
Print_All_Data_LinkList(copy_one, print, ClassicLinkList_OPEN);
```

```
    //interval
    doChangetoTargetIntervalDatainLinkList(copy_one, 0, 2, dochange);
    Print_All_Data_LinkList(copy_one, print, ClassicLinkList_OPEN);

    //all
    doChangetoAllDatainLinkList(copy_one, dochange);
    Print_All_Data_LinkList(copy_one, print, ClassicLinkList_OPEN);

    //check if in
    Classic_DataList* checkerlist = UpdateStaticArray2ClassicLinkList(insertarr, Dyarr_SIGINT,
5);
    int wannafind = -1;
    printf("So is it null?");
    if (!isEmptyClassicLinkList(checkerlist)) {
        printf("\nNot null sir!\n");
    }
    if (checkIsLocateinLinkList(checkerlist, &wannafind, loc)) {
        printf("yes, there is a num that is -1\n");
    }
    printf("and it is on the place: %d\n", returnOutDatabyposinLinkList(checkerlist, &wannafind,
loc));
    int wannafind2 = wannafind - 1;
    printf("and also there is -2 too, it is in: %d\n", returnOutDatabyposinLinkList(checkerlist,
&wannafind2, loc));
    printf("I can also make a change in this way directly:\n");
    doChangetoSpecificElementinLinkList(checkerlist, returnOutDatabyposinLinkList(checkerlist,
&wannafind2, loc), dochange);
    Print_All_Data_LinkList(checkerlist, print, ClassicLinkList_OPEN);

    //find bunch
    Position_Stored_Dynamic_ArrayFordyarr* posarr =
Init_A_Postion_Stored_Dynamic_ArrayFordyarr();
    returnAbunchLocationsinLinkList(checkerlist, posarr, &wannafind, loc);
    Show_All_Locations_In_PSDAfor_dyarr(posarr, PSDA_dyarr_OPEN);

    //reverse
    ReverseLinkList(checkerlist);
    Print_All_Data_LinkList(checkerlist, print, ClassicLinkList_OPEN);

    //Sort
    sortClassicLinkListinBubbleSort(checkerlist, comp);
    Print_All_Data_LinkList(checkerlist, print, ClassicLinkList_OPEN);

    //Del
    clearAClassicLinkList(checkerlist);
    Print_All_Data_LinkList(checkerlist, print, ClassicLinkList_OPEN);
}
```