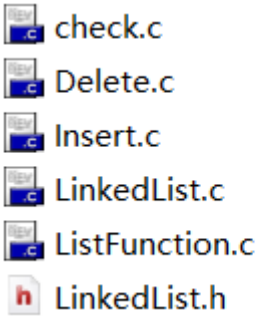
Exercise1

【文件结构】



【源代码】

【LinkedList.h】

```
//链表节点定义
typedef struct Node *PtrToNode;        //指向链表节点的指针
typedef PtrToNode Position;    //指向链表节点的指针
struct Node{
    void *pData;
    int DataType;     //存储数据类型  1:int     2:double     3:char
    PtrToNode Next;
};
```

【LinkedList.c】

```
#include <stdio.h>
#include <stdlib.h>
#include "LinkedList.h"

int main() {
    //创建新链表
    PtrToNode List_1 = CreateLinkedList();
    PtrToNode List_2 = CreateLinkedList();

    //检验链表功能
    CheckInsert(List_1,List_2);
    CheckDelete(List_2);
    CheckMerge(List_1 , List_2);
    CheckReverse(List_1);

    //清除链表
    FreeLinkedList(List_1);

    //通过输出验证清除功能
    printf_s("\n 以下是 List_1 清除后的结果：\n");
    PrintLinkedList(List_1);

    //释放表头
    free(List_1);
    return 0;
}
```

【ListFunction.c】

```
#include "LinkedList.h"
#include <stdio.h>
#include <stdlib.h>

/**
  *@brief 功能：新建链表
  *@return 该函数返回新建链表的哑结点
  *@note 该函数仅仅只是将链表 2 接在链表 1 的后方
*/
PtrToNode CreateLinkedList() {
	PtrToNode Head;
	Head = (PtrToNode)malloc(sizeof(struct Node));
	if (Head == NULL) {
		printf("Out of Space!\n");
	} else {
		Head->Next = NULL;   //初始化 Next 域
		Head->pData = NULL; //初始化 pData 域
	}
	return Head;
}



/**
  *@brief 功能：链表数据销毁
  *@param[in] Head：待销毁链表表头
  *@param[out] Head：原链表表头 Head
*/
void FreeLinkedList(PtrToNode Head) {
	PtrToNode Temp;
	Position P = Head->Next;
	Head->Next = NULL;
	while (P != NULL) {
		Temp = P->Next;
		free(P);
		P = Temp;
	}
}

/**
  *@brief 功能：合并链表
  *@param[in] List_1：链表 1 哑结点
  *@param[in] List_2：链表 2 哑结点
  *@param[out] List_1：合并后链表哑结点
  *@note 该函数仅仅只是将链表 2 接在链表 1 的后方
*/
void MergeLinkedeList(PtrToNode List_1 , PtrToNode List_2) {
	Position P = List_1;
```

```
    while (P->Next != NULL) {
        P = P->Next;
    }
    P->Next = List_2->Next;
    free(List_2); //释放被合并链表 List_2 的哑结点
}

/**
  *@brief 功能：反转链表
  *@param[in] Head：链表哑结点
  *@param[out] Head：新链表哑结点
*/
void ReverseLinkedList(PtrToNode Head) {
    if (Head->Next == NULL || Head->Next->Next == NULL) return Head; //链表空或只有一个元素
    Position P = Head->Next;
    Position New = NULL;
    PtrToNode Temp;
    while (P != NULL) {   //迭代法反转链表
        Temp = P->Next;
        P->Next = New;
        New = P;
        P = Temp;
    }
    Head->Next = New;
}

/**
  *@brief 功能：输出链表每一个元素
  *@param[in] Head：链表哑结点
*/
void PrintLinkedList(PtrToNode Head) {
    int count = 1; //数据计数器
    Position P = Head->Next;
    printf_s("\n");
    printf_s("-------Print LinkedList------\n");
    if (P == NULL) {
        printf_s("\n\n         Linked List Empty!\n\n\n");
        printf_s("-----------------------------\n");
    }
    while (P != NULL) {
        switch (P->DataType) {
            case 1: {
                printf_s("            Data[%d]          \n",count);
                printf_s("DataType: int\n");
                printf_s("Value     : %d\n",*(int*)(P->pData));
                break;
            }
            case 2: {
                printf_s("            Data[%d]          \n" , count);
```

```
                printf_s("DataType: double\n");
                printf_s("Value      : %f\n" , *(double*)(P->pData));
                break;
            }
            case 3: {
                printf_s("            Data[%d]          \n" , count);
                printf_s("DataType: char\n");
                printf_s("Value      : %c\n" , *(char*)(P->pData));
                break;
            }
        }
        P = P->Next;
        count++;
        printf_s("----------------------------\n");
    }
}
```

【Insert.c】

```
#include "LinkedList.h"
#include <stdio.h>
#include <stdlib.h>


/**
  *@brief 功能：在链表哑结点后插入数据
  *@param[in] Head：链表哑结点
  *@param[in] DataType：插入的数据类型
  *@param[out] Head：插入后链表哑结点
*/
void InsertToHead(PtrToNode Head , int DataType) {
    PtrToNode Temp;
    //申请内存空间
    Temp = (PtrToNode)malloc(sizeof(struct Node));
    if (Temp == NULL) return;
    //数据处理
    Temp->DataType = DataType;
    Temp->pData = malloc(sizeof(Temp->pData));
    switch (DataType) {
        case 1: {
            printf_s("int data = ");
            scanf_s("%d%*c" , Temp->pData);
            break;
        }
        case 2: {
            printf_s("double data = ");
            scanf_s("%lf%*c" , Temp->pData);
            break;
        }
```

```
        case 3: {
            printf_s("char data = ");
            fflush(stdin); //清空输入流缓冲区字符
            scanf_s("%c%*c" , Temp->pData);
            break;
        }
    }
    //链表连接
    Temp->Next = Head->Next;
    Head->Next = Temp;
}

/**
  *@brief 功能：在链表末尾插入数据
  *@param[in] Head：链表哑结点
  *@param[in] DataType：插入的数据类型
  *@param[out] Head：插入后链表哑结点
*/
void InsertToTail(PtrToNode Head , int DataType) {
    Position P = Head;
    PtrToNode Temp;
    //找到链表末尾的位置
    while (P->Next != NULL) {
        P = P->Next;
    }
    //申请内存空间
    Temp = (PtrToNode)malloc(sizeof(struct Node));
    if (Temp == NULL) return;
    //数据处理
    Temp->DataType = DataType;
    Temp->pData = malloc(sizeof(Temp->pData));
    switch (DataType) {
        case 1: {
            printf_s("int data = ");
            scanf_s("%d%*c" , Temp->pData);
            break;
        }
        case 2: {
            printf_s("double data = ");
            scanf_s("%lf%*c" , Temp->pData);
            break;
        }
        case 3: {
            printf_s("char data = ");
            fflush(stdin); //清空输入流缓冲区字符
            scanf_s("%c%*c" , Temp->pData);
            break;
        }
    }
```

```
    //链表连接
    P->Next = Temp;
    Temp->Next = NULL;
}

/**
  *@brief 功能：按照整数型在左，实数型在中间，字符型数据在右边的顺序插入数据
  *@param[in] Head：链表哑结点
  *@param[in] DataType：插入的数据类型
  *@param[out] Head：插入后链表哑结点
*/
void InsertByOrder(PtrToNode Head , int DataType) {
    Position P = Head;
    PtrToNode Temp;
    //申请内存空间
    Temp = (PtrToNode)malloc(sizeof(struct Node));
    if (Temp == NULL) return;
    //数据处理与链表连接
    Temp->DataType = DataType;
    Temp->pData = malloc(sizeof(Temp->pData));
    switch (DataType) {
        case 1: {
            printf_s("int data = ");
            scanf_s("%d%*c" , Temp->pData);
            while (P->Next != NULL && P->Next->DataType == 1) { //找到要插入位置的前驱结点
                P = P->Next;
            }
            break;
        }
        case 2: {
            printf_s("double data = ");
            scanf_s("%lf%*c" , Temp->pData);
            while (P->Next != NULL && (P->Next->DataType == 1 || P->Next->DataType == 2)) { //找到要插入位置的前
驱结点
                P = P->Next;
            }
            break;
        }
        case 3: {
            printf_s("char data = ");
            scanf_s("%c%*c" , Temp->pData);
            while (P->Next != NULL) { //找到要插入位置的前驱结点
                P = P->Next;
            }
            break;
        }
    }
    //链表插入
    Temp->Next = P->Next;
```

```
    P->Next = Temp;
}
```

【Delete.c】

```
#include "LinkedList.h"
#include <stdio.h>
#include <stdlib.h>

/**
  *@brief 功能：删除链表哑结点后的结点
  *@param[in] Head：链表哑结点
  *@param[out] Head：删除后链表哑结点
*/
void DeleteFromHead(PtrToNode Head) {
    PtrToNode Temp = Head->Next;
    if (Head->Next == NULL) return;
    Head->Next = Temp->Next;
    free(Temp);
}

/**
  *@brief 功能：删除链表末尾结点
  *@param[in] Head：链表哑结点
  *@param[out] Head：删除后链表哑结点
*/
void DeleteFromTail(PtrToNode Head) {
    Position P = Head;
    PtrToNode Temp;
    if (P->Next == NULL) return;
    while (P->Next->Next != NULL) { //找到尾结点的前驱结点
        P = P->Next;
    }
    Temp = P->Next; //删除操作
    P->Next = NULL;
    free(Temp);
}

/**
  *@brief 功能：删除链表中某一类型数据
  *@param[in] Head：链表哑结点
  *@param[in] DataType：需要删除的数据类型
  *@param[out] Head：删除后链表哑结点
*/
void DeleleByDatatype(PtrToNode Head , int DataType) {
    Position P = Head;
    PtrToNode Temp;
    while (P->Next != NULL) { //找到符合删除条件结点的前驱结点
        if (P->Next->DataType == DataType) {
            Temp = P->Next;
```

```
                P->Next = Temp->Next;
                free(Temp);
                continue; //删除一个数据后 P 指针不需要继续移动位置
            }
            P = P->Next;
        }
}
```

【Check.c】

```
#include "LinkedList.h"
#include <stdio.h>
#include <stdlib.h>

///@brief : 检验链表插入函数
void CheckInsert(PtrToNode List_1 , PtrToNode List_2) {
    //在链表头部分别插入整型、实数型、字符型数据
    InsertToHead(List_1 , 1);
    InsertToHead(List_1 , 2);
    InsertToHead(List_1 , 3);
    //在链表尾部分别插入整型、实数型、字符型数据
    InsertToTail(List_1 , 1);
    InsertToTail(List_1 , 2);
    InsertToTail(List_1 , 3);
    //通过输出验证插入功能
    printf_s("\n 以下是 List_1 链表的数据：\n");
    PrintLinkedList(List_1);
    //按照整数型在左，实数型在中间，字符型数据在右边的顺序插入数据
    InsertByOrder(List_2 , 3);
    InsertByOrder(List_2 , 1);
    InsertByOrder(List_2 , 3);
    InsertByOrder(List_2 , 2);
    InsertByOrder(List_2 , 2);
    InsertByOrder(List_2 , 3);
    InsertByOrder(List_2 , 1);
    //通过输出验证顺序插入功能
    printf_s("\n 以下是 List_2 链表的数据：\n");
    PrintLinkedList(List_2);
}

///@brief : 检验链表删除函数
void CheckDelete(PtrToNode List_2) {
    //从头结点删除链表数据并通过输出验证
    DeleteFromHead(List_2);
    printf_s("\n 以下是 List_2 链表从头结点删除后的结果：\n");
    PrintLinkedList(List_2);
    //删除链表尾节点数据并通过输出验证
    DeleteFromTail(List_2);
    printf_s("\n 以下是 List_2 链表从尾结点删除后的结果：\n");
    PrintLinkedList(List_2);
```

```
    //删除链表中某一类型数据并通过输出验证
    DeleleByDatatype(List_2 , 3);
    printf_s("\n 以下是 List_2 链表删除字符型数据后的结果：\n");
    PrintLinkedList(List_2);
}

///@brief : 检验合并链表函数
void CheckMerge(PtrToNode List_1 , PtrToNode List_2) {
    MergeLinkedeList(List_1 , List_2);
    printf_s("\n 以下是 List_1 和 List_2 合并后的结果：\n");
    PrintLinkedList(List_1);
}

///@brief : 检验反转链表函数
void CheckReverse(PtrToNode List_1) {
    ReverseLinkedList(List_1);
    printf_s("\n 以下是 List_1 反转后的结果：\n");
    PrintLinkedList(List_1);
}
```

【运行结果截图】

链表 1 数据输入：

```
int data = 2018
double data = 9.22
char data = z
int data = 1702
double data = 32.33
char data = y
```

打印链表 1 数据：前 3 个数据为在表头插入，后三个数据为在尾部插入

```
以下是List_1链表的数据：

-------Print LinkedList------
       Data[1]
DataType: char
Value   : z
------------------------------
       Data[2]
DataType: double
Value   : 9.220000
------------------------------
       Data[3]
DataType: int
Value   : 2018
------------------------------
       Data[4]
DataType: int
Value   : 1702
------------------------------
       Data[5]
DataType: double
Value   : 32.330000
------------------------------
       Data[6]
DataType: char
Value   : y
------------------------------
```

链表 2 数据插入：

```
char data = c
int data = 1727
char data = j
double data = 3.14
double data = 7.0000
char data = s
int data = 9
```

链表 2 数据输出：按照 int 在左，double 在中，char 在右的方式插入

```
以下是List_2链表的数据：

-------Print LinkedList------
       Data[1]
DataType: int
Value   : 1727
----------------------------
       Data[2]
DataType: int
Value   : 9
----------------------------
       Data[3]
DataType: double
Value   : 3.140000
----------------------------
       Data[4]
DataType: double
Value   : 7.000000
----------------------------
       Data[5]
DataType: char
Value   : c
----------------------------
       Data[6]
DataType: char
Value   : j
----------------------------
       Data[7]
DataType: char
Value   : s
----------------------------
```

链表 2 删除头结点数据：

```
以下是List_2链表从头结点删除后的结果：

------Print LinkedList------
      Data[1]
DataType: int
Value   : 9
----------------------------
      Data[2]
DataType: double
Value   : 3.140000
----------------------------
      Data[3]
DataType: double
Value   : 7.000000
----------------------------
      Data[4]
DataType: char
Value   : c
----------------------------
      Data[5]
DataType: char
Value   : j
----------------------------
      Data[6]
DataType: char
Value   : s
----------------------------
```

链表 2 删除尾结点数据：

```
以下是List_2链表从尾结点删除后的结果：

------Print LinkedList------
      Data[1]
DataType: int
Value   : 9
----------------------------
      Data[2]
DataType: double
Value   : 3.140000
----------------------------
      Data[3]
DataType: double
Value   : 7.000000
----------------------------
      Data[4]
DataType: char
Value   : c
----------------------------
      Data[5]
DataType: char
Value   : j
----------------------------
```

链表 2 删除 char 类型后的结果：

```
以下是List_2链表删除字符型数据后的结果：

-------Print LinkedList------
       Data[1]
DataType: int
Value    : 9
-----------------------------
       Data[2]
DataType: double
Value    : 3.140000
-----------------------------
       Data[3]
DataType: double
Value    : 7.000000
-----------------------------
```

链表 1 和 2 合并后的结果：

```
以下是List_1和List_2合并后的结果：

-------Print LinkedList------
       Data[1]
DataType: char
Value    : z
-----------------------------
       Data[2]
DataType: double
Value    : 9.220000
-----------------------------
       Data[3]
DataType: int
Value    : 2018
-----------------------------
       Data[4]
DataType: int
Value    : 1702
-----------------------------
       Data[5]
DataType: double
Value    : 32.330000
-----------------------------
       Data[6]
DataType: char
Value    : y
-----------------------------
       Data[7]
DataType: int
Value    : 9
-----------------------------
       Data[8]
DataType: double
Value    : 3.140000
-----------------------------
       Data[9]
DataType: double
Value    : 7.000000
-----------------------------
```

链表 1 反转后的结果：

```
以下是List_1反转后的结果：

-------Print LinkedList------
       Data[1]
DataType: double
Value   : 7.000000
----------------------------
       Data[2]
DataType: double
Value   : 3.140000
----------------------------
       Data[3]
DataType: int
Value   : 9
----------------------------
       Data[4]
DataType: char
Value   : y
----------------------------
       Data[5]
DataType: double
Value   : 32.330000
----------------------------
       Data[6]
DataType: int
Value   : 1702
----------------------------
       Data[7]
DataType: int
Value   : 2018
----------------------------
       Data[8]
DataType: double
Value   : 9.220000
----------------------------
       Data[9]
DataType: char
Value   : z
----------------------------
```

链表 1 清空数据后的结果

```
以下是List_1清除后的结果：

-------Print LinkedList------


     Linked List Empty!


----------------------------
```

【输出文档】

```
int data = 2018
double data = 9.22
char data = z
int data = 1702
double data = 32.33
char data = y
```

以下是 List_1 链表的数据：

```
-------Print LinkedList------
            Data[1]
DataType: char
Value     : z
------------------------------
            Data[2]
DataType: double
Value     : 9.220000
------------------------------
            Data[3]
DataType: int
Value     : 2018
------------------------------
            Data[4]
DataType: int
Value     : 1702
------------------------------
            Data[5]
DataType: double
Value     : 32.330000
------------------------------
            Data[6]
DataType: char
Value     : y
------------------------------
char data = c
int data = 1727
char data = j
double data = 3.14
double data = 7.0000
char data = s
int data = 9
```

以下是 List_2 链表的数据：

```
-------Print LinkedList------
            Data[1]
DataType: int
Value     : 1727
```

```
------------------------------
        Data[2]
DataType: int
Value     : 9
------------------------------
        Data[3]
DataType: double
Value     : 3.140000
------------------------------
        Data[4]
DataType: double
Value     : 7.000000
------------------------------
        Data[5]
DataType: char
Value     : c
------------------------------
        Data[6]
DataType: char
Value     : j
------------------------------
        Data[7]
DataType: char
Value     : s
------------------------------
```

以下是 List_2 链表从头结点删除后的结果：

```
-------Print LinkedList------
        Data[1]
DataType: int
Value     : 9
------------------------------
        Data[2]
DataType: double
Value     : 3.140000
------------------------------
        Data[3]
DataType: double
Value     : 7.000000
------------------------------
        Data[4]
DataType: char
Value     : c
------------------------------
        Data[5]
DataType: char
Value     : j
------------------------------
```

Data[6]
DataType: char
Value : s

以下是 List_2 链表从尾结点删除后的结果：

-------Print LinkedList------
Data[1]
DataType: int
Value : 9

Data[2]
DataType: double
Value : 3.140000

Data[3]
DataType: double
Value : 7.000000

Data[4]
DataType: char
Value : c

Data[5]
DataType: char
Value : j

以下是 List_2 链表删除字符型数据后的结果：

-------Print LinkedList------
Data[1]
DataType: int
Value : 9

Data[2]
DataType: double
Value : 3.140000

Data[3]
DataType: double
Value : 7.000000

以下是 List_1 和 List_2 合并后的结果：

-------Print LinkedList------
Data[1]

DataType: char
Value : z

Data[2]
DataType: double
Value : 9.220000

Data[3]
DataType: int
Value : 2018

Data[4]
DataType: int
Value : 1702

Data[5]
DataType: double
Value : 32.330000

Data[6]
DataType: char
Value : y

Data[7]
DataType: int
Value : 9

Data[8]
DataType: double
Value : 3.140000

Data[9]
DataType: double
Value : 7.000000

以下是 List_1 反转后的结果：

-------Print LinkedList------
Data[1]
DataType: double
Value : 7.000000

Data[2]
DataType: double
Value : 3.140000

Data[3]
DataType: int

Value : 9

Data[4]
DataType: char
Value : y

Data[5]
DataType: double
Value : 32.330000

Data[6]
DataType: int
Value : 1702

Data[7]
DataType: int
Value : 2018

Data[8]
DataType: double
Value : 9.220000

Data[9]
DataType: char
Value : z

以下是 List_1 清除后的结果：

-------Print LinkedList------

Linked List Empty!
