1. 线性单向链表

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

// 只用修改此处就可以实现不同的链表
typedef int ElemType;

// 线性单向链表
typedef struct LNode
{
    ElemType data;
    struct LNode *next;
} LNode, *LinkList;

/**
 * @brief 创建先进先出链表，即队列，顺序插入建立链表
 * @param {ElemType} dataList
 * @return {LinkList}
 */
LinkList createQueue(ElemType *dataList)
{
    LinkList headp = (LinkList)malloc(sizeof(LNode));
    LinkList p = headp;
    for (int i = 0; dataList[i]; i++)
    {
        p->next = (LinkList)malloc(sizeof(LNode));
        p = p->next;
        p->data = dataList[i];
    }
    p->next = NULL;
    return headp;
}

/**
 * @brief 创建先进后出链表，即栈，首插法建立链表
 * @param {ElemType} *dataList
 * @return {LinkList}
 */
LinkList createStack(ElemType *dataList)
{
    LinkList headp = (LinkList)malloc(sizeof(LNode));
    headp->next = NULL;
    for (int i = 0; dataList[i]; i++)
    {
        LinkList temp = (LinkList)malloc(sizeof(LNode));
        temp->data = dataList[i];
        temp->next = headp->next;
        headp->next = temp;
    }
    return headp;
}
```

```
void destoryLinkList(LinkList head)
{
    if (head == NULL)
        return;
    while (head)
    {
        LinkList temp = head->next;
        free(head);
        head = temp;
    }
}
```

2. 线性双向链表：

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

// 只用修改此处就可以实现不同的链表
typedef int ElemType;

// 线性双向链表
typedef struct LNode
{
    ElemType data;
    struct LNode *next, *prior;
} LNode, *LinkList;

/**
 * @brief 创建先进先出链表，即队列，顺序插入建立链表
 * @param {ElemType} dataList
 * @return {LinkList}
 */
LinkList createQueue(ElemType *dataList)
{
    LinkList headp = (LinkList)malloc(sizeof(LNode));
    LinkList ahead = headp, behind;
    for (int i = 0; dataList[i]; i++)
    {
        ahead->next = (LinkList)malloc(sizeof(LNode));
        behind = ahead;
        ahead = ahead->next;
        ahead->prior = behind;
        ahead->data = dataList[i];
    }
    ahead->next = NULL;
    return headp;
}

/**
 * @brief 创建先进后出链表，即栈，首插法建立链表
 * @param {ElemType} *dataList
 * @return {LinkList}
 */
LinkList createStack(ElemType *dataList)
```

```
{
    LinkList headp = (LinkList)malloc(sizeof(LNode));
    headp->next = NULL;
    for (int i = 0; dataList[i]; i++)
    {
        LinkList temp = (LinkList)malloc(sizeof(LNode));
        temp->data = dataList[i];
        temp->prior = headp;
        headp->next->prior = temp;
        temp->next = headp->next;
        headp->next = temp;
    }
    return headp;
}

void destoryLinkList(LinkList head)
{
    if (head == NULL)
        return;
    while (head)
    {
        LinkList temp = head->next;
        free(head);
        head = temp;
    }
}
```

3. 线性循环单向链表

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

// 只用修改此处就可以实现不同的链表
typedef int ElemType;

// 线性单向链表
typedef struct LNode
{
    ElemType data;
    struct LNode *next;
} LNode, *LinkList;

/**
 * @brief 创建先进先出链表，即队列，顺序插入建立链表
 * @param {ElemType} dataList
 * @return {LinkList}
 */
LinkList createQueue(ElemType *dataList)
{
    LinkList headp = (LinkList)malloc(sizeof(LNode));
    LinkList p = headp;
    for (int i = 0; dataList[i]; i++)
    {
        p->next = (LinkList)malloc(sizeof(LNode));
        p = p->next;
```

```
        p->data = dataList[i];
    }
    p->next = headp; // 相较于非循环链表变化之处
    return headp;
}

/**
 * @brief 创建先进后出链表，即栈，首插法建立链表
 * @param {ElemType} *dataList
 * @return {LinkList}
 */
LinkList createStack(ElemType *dataList)
{
    LinkList headp = (LinkList)malloc(sizeof(LNode));
    headp->next = headp; // 相较于非循环链表变化之处
    for (int i = 0; dataList[i]; i++)
    {
        LinkList temp = (LinkList)malloc(sizeof(LNode));
        temp->data = dataList[i];
        temp->next = headp->next;
        headp->next = temp;
    }
    return headp;
}

void destoryLinkList(LinkList head) // 相较于非循环链表变化之处
{
    if (head == NULL)
        return;
    LinkList p = head; // 断链
    head = head->next;
    p->next = NULL;
    while (head)
    {
        LinkList temp = head->next;
        free(head);
        head = temp;
    }
}
```

4. 线性循环双向链表

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

// 只用修改此处就可以实现不同的链表
typedef int ElemType;

// 线性双向链表
typedef struct LNode
{
    ElemType data;
    struct LNode *next, *prior;
} LNode, *LinkList;
```

```
/**
 * @brief 创建先进先出链表，即队列，顺序插入建立链表
 * @param {ElemType} dataList
 * @return {LinkList}
 */
LinkList createQueue(ElemType *dataList)
{
    LinkList headp = (LinkList)malloc(sizeof(LNode));
    LinkList ahead = headp, behind;
    for (int i = 0; dataList[i]; i++)
    {
        ahead->next = (LinkList)malloc(sizeof(LNode));
        behind = ahead;
        ahead = ahead->next;
        ahead->prior = behind;
        ahead->data = dataList[i];
    }
    ahead->next = headp;  // 相较于非循环链表变化之处
    headp->prior = ahead; // 相较于非循环链表变化之处
    return headp;
}

/**
 * @brief 创建先进后出链表，即栈，首插法建立链表
 * @param {ElemType} *dataList
 * @return {LinkList}
 */
LinkList createStack(ElemType *dataList)
{
    LinkList headp = (LinkList)malloc(sizeof(LNode));
    headp->next = headp; // 相较于非循环链表变化之处
    for (int i = 0; dataList[i]; i++)
    {
        LinkList temp = (LinkList)malloc(sizeof(LNode));
        temp->data = dataList[i];
        temp->prior = headp;
        headp->next->prior = temp;
        temp->next = headp->next;
        headp->next = temp;
    }
    return headp;
}

void destoryLinkList(LinkList head)
{
    if (head == NULL)
        return;
    LinkList p = head; // 断链
    head = head->next;
    head->prior = NULL;
    p->next = NULL;
    while (head)
    {
        LinkList temp = head->next;
        free(head);
```

```
            head = temp;
        }
    }
```