1. 单链表的结构定义

```
#include <iostream>
using namespace std;

typedef struct LNode{
    int data;
    struct LNode *next;
} LNode;
```

2. 单链表的基本操作

2.1 初始化

```
int init(LNode *&L){
    L = (LNode *) malloc(sizeof(LNode *));
    L -> next = NULL;
}
```

2.2 创建单链表 (头插法)

```
void crateListF(LNode *&C, int a[], int n){
    LNode *s;

    C = (LNode *)malloc(sizeof(LNode *));
    C-> next = NULL;

    for(int i=0; i<n; i++){
        s = (LNode *) malloc(sizeof(LNode *));
        s->data = a[i];

        //s 所指新结点的指针域 next 指向 C 中的开始结点。
```

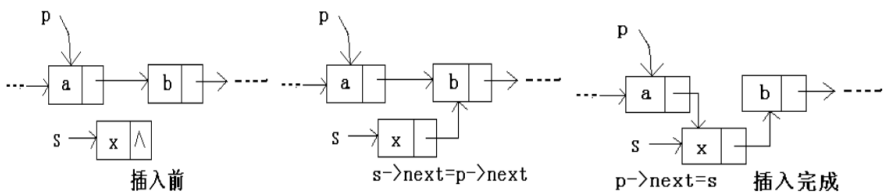
```
        s->next = C->next;

        //头结点的指针域 next 指向 s 结点，使得 s 成为了新的开始结点。
        C->next = s;
    }
}
```

2.3 创建单链表 (尾插法)

```
void createListR(LNode *&C, int a[], int n){
    //s用来指向新申请的结点，r始终指向C的终端结点。s
    LNode *s,*r;

    //申请 C 的头结点空间。
    C = (LNode *) malloc(sizeof(LNode *));
    C->next = NULL;
    r = C;

    for(int i=0; i<n; i++){
        // 指向新申请的结点
        s = (LNode *)malloc(sizeof(LNode *));
        s->data = a[i];
        // 不赋初值的话会出现 Segmentation fault
        s->next = NULL;

        //用 r 来接纳新结点。
        r->next = s;

        //r 指向终端结，点以便于接纳下一个到来的结点。
        r = r->next;
    }
    r->next = NULL;
}
```

2.4 指定位置插入节点

链表非空的情况下，在 $1 \leq p \leq length$ 的位置插入节点



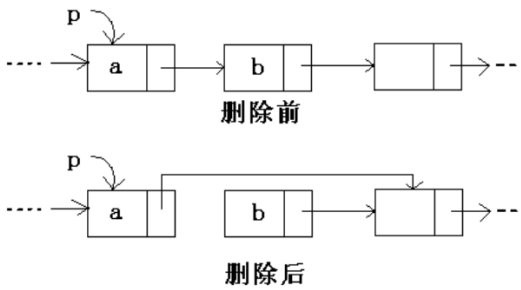
2.10 插入操作过程图

```
int insertLinkedList(LNode *&L, int p, int data){
    LNode *q = L, *s;
    // 当前遍历到第 i 个节点
    int i = 0;
    // 找到第 p-1 个节点
    while(p!=NULL && i<p-1){
        q = q->next;
        i++;
```

```
    }
    // 插入节点
    if(q != NULL){
        s = (LNode *) malloc(sizeof(LNode *));
        s->data = data;
        s->next = q->next;
        q->next = s;
        return 1;
    }else{
        return 0;
    }
}
```

2.5 删除指定位置的节点

将单链表的第 i 个结点删去，**必须先 在单链表中找到第 $i-1$ 个结点**，再删除其后继结点。



2.11 删除操作过程图

```
int delElem(LNode *&L, int p, int &data){
    if(L->next==NULL){
        return 0;
    }
    int i=0;
    LNode *q = L, *s;

    // 找到第 p-1 个节点
    while(q->next!=NULL && i<p-1){
        q = q->next;
        i++;
    }

    // 第 p 个节点为空，返回错误
    if(q->next == NULL){
        return 0;
    }

    // 删除第 p 个节点
    s = q->next;
    q->next = s->next;
    data = s->data;
    free(s);
    return 1;
}
```

2.6 获取指定位置的节点

```
int findElem(LNode *&L, int p, int &data) {
    // 链表为空
    if (L->next==NULL || p<1) {
        return 0;
    }
    int i = 0;
    LNode *q = L, *s;

    // 找到第 p 个节点
    while (q->next!=NULL && i<p) {
        q = q->next;
        i++;
    }
    if(q!=NULL){
        data = q->data;
        return 1;
    }else{
        return 0;
    }
}
```

2.7 获取长度

```
int getLength(LNode *&L){
    int length=0;
    // 指向第一个节点
    LNode *p = L->next;
    while(p!=NULL){
        length++;
        p = p->next;
    }
    return length;
}
```

2.8 打印单链表

```
void printLinkedListH(char* name, LNode *a){
    if(a==NULL){
        printf("单链表为空!!\n");
        return;
    }
    LNode *tmp;
    tmp = a->next;
    printf("单链表 %s : [", name);
    while(tmp){
        printf(" %d ",tmp->data);
        tmp = tmp->next;
    }
    printf("]\n");
}
```

3. 例题

- A和B是两个单链表(带表头结点)，其中元素递增有序。设计一个算法将A 和 B 归并成一个按元素递增有序的链表 C，C 由 A 和 B 中的结点组成。

```
void mergeR(LNode *A, LNode *B, LNode *&C){
    LNode *p = A->next;
    LNode *q = B->next;
    LNode *r;

    C = (LNode*) malloc(sizeof(LNode *));
    C->next = NULL;
    r = C;

    // A、B 同时不为空时，比较数据域大小
    while(p!=NULL & q!=NULL){
        if(p->data > q->data){
            r->next = q;
            q = q->next;
        }else{
            r->next = p;
            p = p->next;
        }
        r = r->next;
    }
    r->next=NULL;

    // A不为空则将A剩余的节点拼接到C，反之亦然
    if(p != NULL)
        r->next = p;
    if(q != NULL)
        r->next = q;
}
```

- 查找链表 C(带头结点)中是否存在一个值为 x 的结点，存在就删除之并返回 1，否则返回 0。

```
int SearchAndDelete(LNode *&C,int x){
    if(C==NULL){
        return 0;
    }

    // 查找部分
    LNode *p, *q;
    p = C;
    while(p->next != NULL){
        if(p->next->data == x){
            break;
        }
        p = p->next;
    }

    // 删除部分
    if(p->next == NULL){
        return 0;
    }else{
        q = p->next;
        p->next = p->next->next;
```

```
        free(q);
        return 1;
    }
}
```