

## 数据结构实验报告

张恒硕      2212266      智科 2 班

实验目标：顺序表，单链表，双向链表

### 一、顺序表

实验原理：

顺序表是一种基于数组的数据结构，使用一段连续的内存空间来存储数据元素。

代码：

```
#include <iostream>
#define MAXSIZE 100 // 定义最大长度
using namespace std;
class List {
private:
    int data[MAXSIZE];
    int length;
public:
    List() { //构造函数
        length = 0;
    }
    ~List() {} //析构函数
    bool Empty() { //判断是否为空
        return length == 0;
    }
    bool Full() { //判断是否已满
        return length == MAXSIZE;
    }
    int Length() { //返回当前长度
        return length;
    }
    void clear() { //清空
        length = 0;
        cout << "已清空";
    }
    bool insert(int index, int val) { //在指定位置插入元素
        if (index < 1 || index > length + 1 || Full()) {
            return false;
        }
        for (int i = length; i >= index; i--) {
            data[i] = data[i - 1];
        }
        data[index - 1] = val;
```

```

        length++;
        return true;
    }
    bool remove(int index) { //删除指定位置的元素
        if (index < 1 || index > length || Empty()) {
            return false;
        }
        cout << data[index - 1] << endl;
        for (int i = index; i < length; i++) {
            data[i - 1] = data[i];
        }
        length--;
        return true;
    }
    int setElem(int index, int val) { //设置指定位置的元素
        if (index < 1 || index > length || Empty()) {
            return -1;
        }
        data[index - 1] = val;
    }
    int getElem(int index) { //获取指定位置的元素
        if (index < 1 || index > length || Empty()) {
            return -1;
        }
        return data[index - 1];
    }
    void printList() { //打印所有元素
        for (int i = 0; i < length; i++) {
            cout << data[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    List list;
    int n, a;
    cout << "请输入测试项数目" << endl;
    cin >> n; //输入: 3
    cout << "请按顺序输入各测试项" << endl;
    for (int i = 1; i <= n; i++) { //输入: 1 2 3
        cin >> a;
        list.insert(i, a);
    }
    list.printList(); //输出: 1 2 3
}

```

```

    list.remove(2); //输出: 2
    list.printList(); //输出: 1 3
    cout << list.getElem(2) << endl; //输出: 3
    list.setElem(2, 4);
    cout << list.getElem(2) << endl; //输出: 4
    cout << list.Length() << endl; //输出: 2
    list.clear();
    list.printList(); //输出: 已清空
    return 0;
}

```

输入样例:

```

3
1 2 3

```

运行结果:

```

Microsoft Visual Studio
请输入测试项数目
3
请按顺序输入各测试项
1 2 3
1 2 3
2
1 3
3
4
2
已清空

C:\Users\zhs20\Desktop\Study\数据结构 (刘进超, 大二上)\数据结构\顺序表\x64\Debug\顺序表.exe (进程 14812)已退出, 代码为 0。
按任意键关闭此窗口...

```

实现操作:

构造函数

析构函数

判断是否为空

判断是否已满

清空

在指定位置插入元素

删除指定位置的元素

设置指定位置的元素

获取指定位置的元素

打印所有元素

## 二、单链表

实验原理：

单链表是一种线性数据结构，其中每个元素都包含一个链接到下一个元素的引用，可通过创建节点类和链表类来实现。节点类表示链表中的元素，它包含一个存储节点值的数据成员和一个指向下一个节点的指针。链表类包含指向链表头节点的指针和一些管理链表（如改变节点之间的指针关系等）的操作。

代码：

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int x) : data(x), next(NULL) {} //构造函数
};
class List {
private:
    Node* head;
public:
    List() { //构造函数
        head = NULL;
    }
    ~List() {} //析构函数
    void clear() { //清空
        Node* curr = head;
        while (curr != NULL) {
            Node* next = curr->next;
            delete curr;
            curr = next;
        }
        head = NULL;
        cout << "已清空";
    }
    bool Empty() { //判断是否为空
        if (head == NULL) {
            return 1;
        }
        return 0;
    }
    Node* search(int val) { //查找元素
        Node* current = head;
        while (current != NULL && current->data != val) {
            current = current->next;
        }
    }
};
```

```

    }
    return current;
}

void update(int oldVal, int newVal) { //修改元素
    Node* node = search(oldVal);
    if (node != NULL) {
        node->data = newVal;
    }
}

void insert(int val) { //插入元素
    Node* node = new Node(val);
    if (head == NULL) {
        head = node;
    }
    else {
        Node* curr = head;
        while (curr->next != NULL) {
            curr = curr->next;
        }
        curr->next = node;
    }
}

void remove(int val) { //删除元素
    if (head == NULL) {
        return;
    }
    if (head->data == val) {
        Node* temp = head;
        head = head->next;
        delete temp;
        cout << val << endl;
        return;
    }
    Node* prev = head;
    Node* curr = head->next;
    while (curr != NULL && curr->data != val) {
        prev = curr;
        curr = curr->next;
    }
    if (curr != NULL) {
        Node* temp = curr;
        prev->next = curr->next;
        cout << val << endl;
        delete temp;
    }
}

```

```

    }
}

void print() { //打印所有元素
    Node* curr = head;
    while (curr != NULL) {
        cout << curr->data << " ";
        curr = curr->next;
    }
    cout << endl;
}

void length() { //返回当前长度
    int len = 0;
    Node* curr = head;
    while (curr != NULL) {
        len++;
        curr = curr->next;
    }
    cout << len << endl;
}

};

int main() {
    List list;
    int n, a;
    cout << "请输入测试项数目" << endl;
    cin >> n; //输入: 4
    cout << "请按顺序输入各测试项" << endl;
    for (int i = 1; i <= n; i++) { //输入: 1 2 3 4
        cin >> a;
        list.insert(a);
    }

    list.print(); //输出: 1 2 3 4
    list.length(); //输出: 4
    list.remove(3); //输出: 3
    list.print(); //输出: 1 2 4
    list.update(4, 3);
    list.print(); //输出: 1 2 3
    list.clear();
    list.print(); //输出: 已清空
    return 0;
}

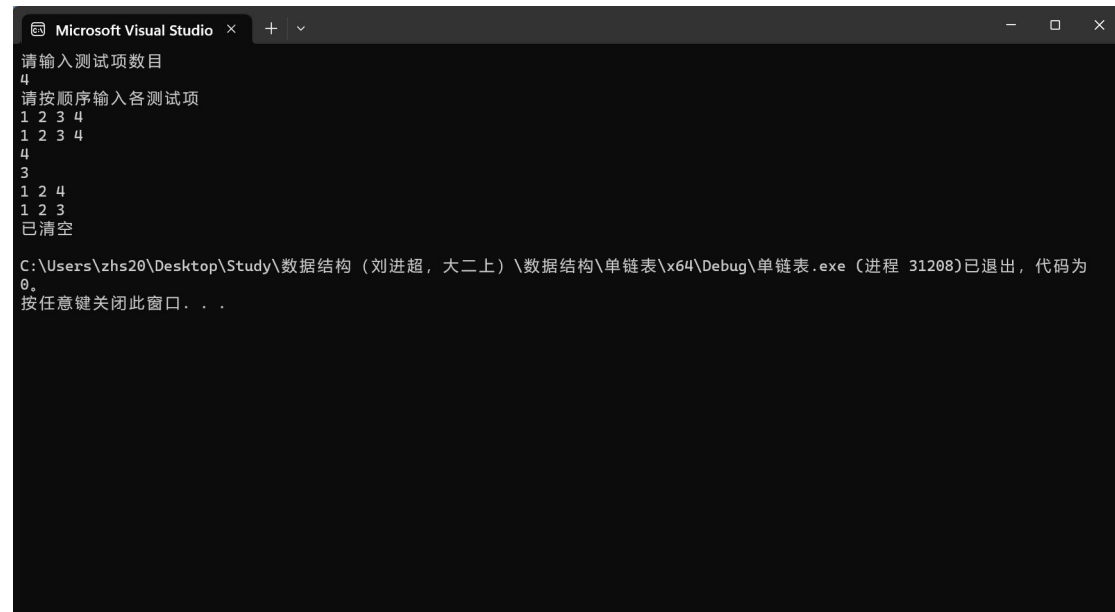
```

输入样例:

4

1 2 3 4

运行结果：



```
Microsoft Visual Studio × + -
请输入测试项数目
4
请按顺序输入各测试项
1 2 3 4
1 2 3 4
4
3
1 2 4
1 2 3
已清空

C:\Users\zhs20\Desktop\Study\数据结构（刘进超，大二上）\数据结构\单链表\x64\Debug\单链表.exe（进程 31208）已退出，代码为 0。
按任意键关闭此窗口...
```

实现操作：

构造函数

析构函数

清空

判断是否为空

查找元素

修改元素

插入元素

删除元素

打印所有元素

返回当前长度

### 三、双向链表

实验原理：

双向链表是一种链表，它每个数据结点中都有两个指针，分别指向其直接后继和直接前驱（头部节点没有前一个节点，尾部节点没有后一个节点）。

代码：

```
#include <iostream>
using namespace std;
class Node {
public:
    int data;
    Node* prev;
    Node* next;
```

```

Node(int val) { //构造函数
    data = val;
    prev = NULL;
    next = NULL;
}
};

class List {
public:
    Node* head;
    Node* tail;
    List() { //构造函数
        head = NULL;
        tail = NULL;
    }
    ~List() {} //析构函数
    void clear() { //清空
        Node* curr = head;
        while (curr != NULL) {
            Node* next = curr->next;
            delete curr;
            curr = next;
        }
        head = NULL;
        cout << "已清空";
    }
    bool Empty() { //判断是否为空
        if (head == NULL) {
            return 1;
        }
        return 0;
    }
    Node* search(int val) { //查找元素
        Node* current = head;
        while (current != NULL && current->data != val) {
            current = current->next;
        }
        return current;
    }
    void update(int oldVal, int newVal) { //修改元素
        Node* node = search(oldVal);
        if (node != NULL) {
            node->data = newVal;
        }
    }
}

```



```

void insert(int val) { //插入元素
    Node* newNode = new Node(val);
    if (head == NULL) {
        head = newNode;
        tail = newNode;
    }
    else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
}

void prepend(int val) { //在头部添加元素
    Node* newNode = new Node(val);
    if (head == NULL) {
        head = newNode;
        tail = newNode;
    }
    else {
        head->prev = newNode;
        newNode->next = head;
        head = newNode;
    }
}

void delete_(Node* node) { //删除元素
    if (node == head) {
        head = node->next;
        if (head != NULL) {
            head->prev = NULL;
        }
        else {
            tail = NULL;
        }
    }
    else if (node == tail) {
        tail = node->prev;
        if (tail != NULL) {
            tail->next = NULL;
        }
        else {
            head = NULL;
        }
    }
    else {

```

```

        node->prev->next = node->next;
        node->next->prev = node->prev;
    }
    delete node;
}

void print() { //打印所有元素
    Node* curr = head;
    while (curr != NULL) {
        cout << curr->data << " ";
        curr = curr->next;
    }
    cout << endl;
}

};

int main() {
    List list;
    int n, a;
    cout << "请输入测试项数目" << endl;
    cin >> n; //输入: 3
    cout << "请按顺序输入各测试项" << endl;
    for (int i = 1; i <= n; i++) { //输入: 1 2 3
        cin >> a;
        list.insert(a);
    }
    list.prepend(0);
    list.print(); //输出: 0 1 2 3
    list.delete_(list.head->next);
    list.print(); //输出: 0 2 3
    list.update(0, 1);
    list.print(); //输出: 1 2 3
    list.clear();
    list.print(); //输出: 已清空
    return 0;
}

```

输入样例:

3

1 2 3

运行结果:

```

Microsoft Visual Studio
请输入测试项数目
3
请按顺序输入各测试项
1 2 3
0 1 2 3
0 2 3
1 2 3
已清空

C:\Users\zhs20\Desktop\Study\数据结构（刘进超，大二上）\数据结构\双向链表\x64\Debug\双向链表.exe（进程 18200）已退出，代
码为 0。
按任意键关闭此窗口。 . . |

```

#### 四、比较

	空间效率	时间效率
顺序表	存储每个数据元素时没有浪费空间。	访问：在顺序表中是直接定位的，所需的操作次数是 $O(1)$ ； 插入和删除：平均时间和最差时间均为 $O(n)$ ；
链表	每个结点上除存储数据元素外，还要存放一个指针，这个指针一般称为结构性开销。	访问：单链表不能直接访问上述元素，只能从表头开始逐个查找，直到找到第 $i$ 个结点为止。平均时间和最差时间均为 $O(n)$ ； 插入和删除：链表的insert和remove操作所需时间仅为 $O(1)$ ；

（截自 ppt）

顺序表：

优点：

- 1、在内存中是连续存储的，因此可以直接通过下标访问元素，时间复杂度为  $O(1)$ 。
- 2、空间利用率高，每个元素只需要存储一次。

缺点：

- 1、插入和删除操作需移动大量元素，时间复杂度为  $O(n)$ 。
- 2、大小固定，不能动态扩展。

单链表：

优点：

- 1、在内存中可以非连续存储，因此可以动态扩展。

2、插入和删除操作只需要修改指针，时间复杂度为  $O(1)$ 。

缺点：

1、只能通过头指针访问元素，时间复杂度为  $O(n)$ 。

2、空间利用率低，每个元素需要额外存储一个指针。

双向链表：

优点：

1、在内存中可以非连续存储，因此可以动态扩展。

2、插入和删除操作只需要修改指针，时间复杂度为  $O(1)$ 。

3、可以通过前驱指针和后继指针访问元素，时间复杂度为  $O(1)$ 。

缺点：

1、空间利用率低，每个元素需要额外存储两个指针。

2、头尾插入和删除操作需要处理特殊情况。