基础数据结构

线性表

线性表是数据结构中最简单也是最常用的一种结构

线性表由相同数据类型的 n 个数据元素组成的有限序列

第一个数据元素是唯一的"第一个"数据元素，又称为表头元素，同理最后一个元素是唯一的"最后一个"数据元素，又称为表尾元素

顺序表

顺序表时线性表中的一种顺序存储形式。

顺序表在程序中通常用一维数组实现，一堆数组可以是静态分配的，也可以是动态分配的。

静态分配

由于数组的大小和空间是固定的，一旦空间占满，就无法再新增数据，否则会导致数据溢出

动态分配

存储数组的空间在程序执行的过程中会动态调整大小，当空间占满时，可以开辟更大的空间进行存储

核心代码

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

//声明结构体
typedef struct Vector {
    int *data;
    int size, length;
} Vector;

//构造或初始化
void init(Vector *v, int n) {
    v->size = n;
    v->length = 0;
    v->data = (int *)malloc(sizeof(int) * n);
    return ;
}

//扩容
void expand(Vector *v) {
    int *p = (int *)realloc(v->data, sizeof(int) * 2 * b->size);
    if(p == NULL) return 0;
```

```
    v->data = p;
    v->size *= 2;
    return 1;
}

//插入
int insert(Vector *v, int ind, int value) {
    if(ind < 0 || ind > v->length) {
        return 0;
    }
    if(v->length >= v->size) {
        expand(v);
        return 0;
    }
    for(int i = v->length; i > ind; --i) {
        v->data[i] = v->data[i - 1];
    }
    v->data[ind] = value;
    v->length++;
    return 1;
}

//删除
int erase(Vector *v, int ind) {
    if(v->length == 0) {
        return 0;
    }
    if(ind < 0 || ind >= v->length) {
        return 0;
    }
    for(int i = ind + 1; i < v->length; i++)
    {
        v->data[i - 1] = v->data[i];
    }
    v->length -= 1;
    return 1;
}

//查找
int search(Vector *v, int value) {
    for(int i = 0; i < v->length; ++ i) {
        if(v->data[i] == value) {
            return 1;
        }
    }
    return 0;
}

//遍历
void output(Vector *v) {
    for(int i = 0; i < v->length; i++) {
        printf("%d ", v->data[i]);
    }
```

```
    printf("\n");
    return ;
}

//清除
void clear(Vector *v) {
    if(v == NULL) return ;
    free(v->data);
    free(v);
    return;
}

int main() {
   Vector *a = (Vector *)malloc(sizeof(Vector));
    init(a, 20);
    int t, n, x, y, s;
    scanf("%d", &t);
    while(t--) {
        scanf("%d", &n);
        if(n == 1) {
            scanf("%d %d", &x, &y);
            s = insert(a, x, y);
            if(s == 1) {
                printf("success\n");
            }
            if(s == 0) {
                printf("failed\n");
            }
        }
        if(n == 2) {
            scanf("%d", &x);
            s = erase(a, x);
            if(s == 1) {
                printf("success\n");
            }
            if(s == 0) {
                printf("failed\n");
            }
        }
        if(n == 3) {
            scanf("%d", &x);
            s = search(a, x);
            if(s == 1) {
                printf("success\n");
            }
            if(s == 0) {
                printf("failed\n");
            }
        }
        if(n == 4) {
            output(a);
        }
    }
```

```
    return 0;
}
```

链表

链表是一种常见的基础数据结构，结构体指针在这里得到了充分的利用。链表可以动态的进行存储分配，也就是说，链表是一个功能极为强大的数组，他可以在节点中定义多种数据类型，还可以根据需要随意增添，删除，插入节点

核心代码

```
typedef struct Node{
    int data;
    struct Node *next;
}Node, *LinkedList;

LinkedList insert(LinkedList head, Node *node, int index) {
    if (head == NULL) {
        if (index != 0) {
            printf("failed\n");
            return head;
        }
        head = node;
        printf("success\n");
        return head;
    }
    if (index == 0) {
        node->next = head;
        head = node;
        printf("success\n");
        return head;
    }
    Node *current_node = head;
    int count = 0;
    while (current_node->next != NULL && count < index - 1) {
        current_node = current_node->next;
        count++;
    }
    if (count == index - 1) {
        node->next = current_node->next;
        current_node->next = node;
        printf("success\n");
        return head;
    }
    printf("failed\n");
    return head;
}

void output(LinkedList head) {
    if (head == NULL) {
```

```
        return;
    }
    Node *current_node = head;
    while (current_node != NULL) {
        printf("%d ", current_node->data);
        current_node = current_node->next;
    }
    printf("\n");
}

LinkedList delete_node(LinkedList head, int index) {
    if(head == NULL) {
        printf("failed\n");
        return head;
    }
    Node *current_node = head;
    int count = 0;
    if(index == 0) {
        head = head->next;
        free(current_node);
        printf("success\n");
        return head;
    }
    while (current_node->next != NULL && count < index - 1) {
        current_node = current_node->next;
        count++;
    }
    if(count == index - 1 && current_node->next != NULL) {
        Node *delete_node = current_node->next;
        current_node->next = delete_node->next;
        free(delete_node);
        printf("success\n");
        return head;
    }
    printf("failed\n");
    return head;
}

LinkedList reverse(LinkedList head) {
    if(head == NULL) {
        return head;
    }
    Node *next_node, *current_node;
    current_node = head->next;
    head->next = NULL;
    while(current_node != NULL){
        next_node = current_node->next;
        current_node->next = head;
        head = current_node;
        current_node = next_node;
    }
    return head;
}
```

```
void clear(LinkedList head) {
    Node *current_node = head;
    while (current_node != NULL) {
        Node *delete_node = current_node;
        current_node = current_node->next;
        free(delete_node);
    }
}
```