# 第十四周实验报告

1. 预习结构体，使用结构体编写一个链表，存储整型数据，使用结构体指针\*Head来引导，链表末尾\*Next指针为Null。编写函数，对链表元素进行遍历输出。

 - 源码

文本

描述已自动生成

文本

描述已自动生成

2. 对于上一个练习的链表，编写函数用以在特定位置（第K个元素）插入一个元素，保证链表正常使用。（提示，使用malloc函数分配一个动态空间）

 - 源码

文本

描述已自动生成

3. 对于上一个练习的链表，编写函数用以在特定位置（第K个元素）删除一个元素，保证链表正常使用。（提示，用free去删除一个动态空间）

- 源码

文本

描述已自动生成

- 实验结果

文本

中度可信度描述已自动生成

文本

中度可信度描述已自动生成

图片包含 图形用户界面

描述已自动生成

完整源码

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

    int data;

    struct node \*next;

} node;

// 函数声明

void prt(node \*head);

node \*new(void);

node \*ins(node \*head, int k, int data);

node \*del(node \*head, int k);

int len(node \*head);

int main(int argc, const char \*argv[]) {

    node \*head = NULL;

    printf("请输入数据,输入-1结束输入\n");

    head = new();

    int choice, k, data;

    while (1) {

        printf("\n当前链表长度: %d\n", len(head));

        printf("=== 链表操作菜单 ===\n");

        printf("1. 插入元素\n");

        printf("2. 删除元素\n");

        printf("3. 输出链表\n");

        printf("4. 退出程序\n");

        printf("请选择操作: ");

        scanf("%d", &choice);

        switch (choice) {

        case 1:

            printf("请输入要插入的位置和值，用空格隔开：");

            scanf("%d %d", &k, &data);

            head = ins(head, k, data);

            break;

        case 2:

            if (head == NULL) {

                printf("链表为空！\n");

                break;

            }

            printf("请输入要删除的位置：");

            scanf("%d", &k);

            head = del(head, k);

            break;

        case 3:

            if (head == NULL) {

                printf("链表为空！\n");

            } else {

                printf("链表内容：\n");

                prt(head);

            }

            break;

        case 4:

            // 释放链表内存

            while (head != NULL) {

                node \*temp = head;

                head = head->next;

                free(temp);

            }

            printf("程序已退出\n");

            return 0;

        default:

            printf("无效的选择，请重新输入！\n");

        }

    }

}

node \*new(void) {

    node \*head = NULL;

    node \*tail = NULL;

    int number;

    do {

        scanf("%d", &number);

        if (number != -1) {

            node \*p = (node \*)malloc(sizeof(node));

            if (p == NULL) {

                printf("内存分配失败\n");

                return head;

            }

            p->data = number;

            p->next = NULL;

            if (head == NULL) {

                head = tail = p;

            } else {

                tail->next = p;

                tail = p;

            }

        }

    } while (number != -1);

    return head;

}

int len(node \*head) {

    int length = 0;

    node \*current = head;

    while (current != NULL) {

        length++;

        current = current->next;

    }

    return length;

}

// 输出链表元素的函数

void prt(node \*head) {

    node \*now = head;

    while (now!= NULL) {

        printf("%d\n", now->data);

        now = now->next;

    }

}

// 在指定位置插入元素的函数

node \*ins(node \*head, int k, int data) {

    node \*temp = (node \*)malloc(sizeof(node));

    if (temp == NULL) {

        printf("内存分配失败\n");

        return head;

    }

    temp->data = data;

    if (k == 1) {

        temp->next = head;

        return temp;

    }

    node \*now = head;

    for (int i = 1; i < k - 1 && now!= NULL; i++) {

        now = now->next;

    }

    if (now == NULL) {

        printf("插入位置超出链表长度！\n");

        free(temp);

        return head;

    }

    temp->next = now->next;

    now->next = temp;

    return head;

}

// 在指定位置删除元素的函数

node \*del(node \*head, int k) {

    if (head == NULL) {

        printf("链表为空，无法删除元素！\n");

        return head;

    }

    if (k == 1) {

        node \*temp = head;

        head = head->next;

        free(temp);

        return head;

    }

    node \*prev = head;

    for (int i = 1; i < k - 1 && prev!= NULL; i++) {

        prev = prev->next;

    }

    if (prev == NULL || prev->next == NULL) {

        printf("删除位置超出链表长度！\n");

        return head;

    }

    node \*temp = prev->next;

    prev->next = prev->next->next;

    free(temp);

    return head;

}