

数据结构期末复习

C语言是面向过程编程语言，试试调整一下思路。

完整的代码格式（仅限本课程）

完整的代码需要包含以下部分

- include 导入标准库
- typedef 定义相关结构体
- init()/create() 初始化函数
- add()/insert() 添加节点
- delete()/free() 释放节点
- print() 遍历并打印节点

一般情况下需要的库

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

Q：为什么是这两个库？

- <stdio.h>：主要提供具有输入输出功能的函数。本章主要使用printf()/scanf()，本章不需要但常见的有fread()/fscanf()。
- <stdlib.h>：主要提供通用的工具函数。本章主要使用malloc()/free()，本章不需要但常见的有abs()/exit()。

注意事项

使用“在第i个位置插入、删除节点”这类表述时，第i个位置为从前往后数第i个节点，i从1开始计数，即第1个节点为第一个位置。若描述为“在i位置的节点”，则一般指索引为i的节点，i从0开始计数，索引值为0的节点为第一个节点。

顺序表

- typedef 定义结构体

需要的变量

- 存储数据的容器data[MAXSIZE]
- 顺序表大小size（或者length）

```
typedef struct {  
    int data[MAXSIZE];  
    int size;  
} SeqList;
```

- initSeqList() 初始化函数

函数逻辑

- 将顺序表大小size初始化为0

具体代码实现

```
// 初始化，返回指针类型为顺序表的指针。  
SeqList* initSeqList() {  
    SeqList *list = (SeqList*)malloc(sizeof(SeqList));  
    list->size = 0;  
    return list;  
}
```

在主函数中的用法

```
int main() {  
    SeqList *list = initSeqList();  
    return 0;  
}
```

附课件相关代码原文

```
void initial_List(sequenlist *L) {  
    L->last = 0;  
}
```

此时主函数的用法变为

```
int main() {
    sequenlist L;
    initial_List(sequenlist *L);
    return 0;
}
```

注意：课件上的初始化函数void initial_List(sequenlist *L)并不返回任何值，而是传入一个已经定义为顺序表的变量L的地址。这种情况下在main()函数中需要定义新的变量类型。此处仅展示代码区别。

- addNode() 添加元素函数

函数逻辑

- 用if判断顺序表size是否超过上限MAXSIZE，如果没有则继续进行添加操作，超过则报错。

```
// 添加元素到顺序表
void addNode(SeqList *list, int element) {
    if (list->size < MAXSIZE) {
        list->data[list->size] = element;
        list->size++;
    } else {
        printf("顺序表已满，无法添加元素。\\n");
    }
}
```

- insertNode() 插入元素函数

函数逻辑

- 判断插入位置是否有效
- 判断顺序表是否已满
- 将插入位置后的元素后移
- 插入新元素

```
// 在第i个位置插入元素到顺序表
void insertNode(SeqList *list, int i, int x) {
    if (i < 0 || i > list->size) {
        printf("插入位置无效。\\n");
        return;
    }
    if (list->size >= MAXSIZE) {
        printf("顺序表已满，无法插入元素。\\n");
        return;
    }
    for (int j = list->size; j > i - 1; j--) {
        list->data[j] = list->data[j - 1];
    }
    list->data[i - 1] = x;
    list->size++;
}
```

- deleteNode() 删除元素函数

函数逻辑

- 判断删除位置是否有效
- 将删除位置后的元素逐个前移，只需前移即可覆盖要删除的元素，不需要“删除操作”
- 减少顺序表size的大小

```
// 删除第i个位置的元素
void deleteNode(SeqList *list, int i) {
    if (i < 0 || i >= list->size) {
        printf("删除位置无效。\\n");
        return;
    }
    for (int j = i - 1; j < list->size - 1; j++) {
        list->data[j] = list->data[j + 1];
    }
    list->size--;
}
```

- printSeqList() 遍历并打印顺序表

```
// 打印顺序表
void printSeqList(SeqList *list) {
    for (int i = 0; i < list->size; i++) {
        printf("%d ", list->data[i]);
    }
    printf("\n");
}
```

- freeSeqList() 释放顺序表内存

```
// 释放顺序表
void freeSeqList(SeqList *list) {
    free(list);
}
```

链表

- typedef 定义节点结构体

需要的变量

- 存储数据的变量data
- 指向下一个节点的指针next

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

- createNode() 创建新节点函数

函数逻辑

- 分配内存
- 初始化数据和指针

```
// 创建新节点
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

- insertNode() 插入新节点函数

函数逻辑

- 创建新节点
- 如果链表为空，将新节点设为头节点
- 否则，遍历链表到末尾，插入新节点

// 插入新节点到链表末尾

```
void insertNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}
```

- printList() 遍历并打印链表

// 打印链表

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

- freeList() 释放链表内存

```
// 释放链表内存
void freeList(Node* head) {
    Node* nodeToFree;
    while (head != NULL) {
        nodeToFree = head;
        head = head->next;
        free(nodeToFree);
    }
}
```

- createList() 建立链表函数

函数逻辑

- 循环输入数据，调用insertNode()插入节点

```
// 建立链表
Node* createList() {
    Node* head = NULL;
    int inputValue;

    printf("请输入整数，输入单个节点后按enter继续输入，输入-1结束：\n");
    while (1) {
        scanf("%d", &inputValue);
        if (inputValue == -1) {
            break;
        }
        insertNode(&head, inputValue);
    }
    return head;
}
```

在主函数中的用法

```
int main() {
    Node* head = createList();

    printf("建立的链表内容如下：\n");
    printList(head);

    // 释放链表内存
    freeList(head);

    return 0;
}
```

顺序栈

- typedef 定义结构体

需要的变量

- 存储数据的容器data[MAXSIZE]
- 栈顶指针top

```
typedef struct {  
    int data[MAXSIZE];  
    int top;  
} Stack;
```

- initStack() 初始化函数

函数逻辑

- 将栈顶指针top初始化为-1

```
// 初始化栈  
void initStack(Stack *s) {  
    s->top = -1;  
}
```

- isEmpty() 判断栈是否为空

函数逻辑

- 判断栈顶指针top是否为-1

```
// 判断栈是否为空  
int isEmpty(Stack *s) {  
    return s->top == -1;  
}
```

- isFull() 判断栈是否满

函数逻辑

- 判断栈顶指针top是否为MAXSIZE - 1

```
// 判断栈是否满  
int isFull(Stack *s) {  
    return s->top == MAXSIZE - 1;  
}
```

- push() 入栈函数

函数逻辑

- 判断栈是否满
- 如果未满，将元素压入栈顶，并更新栈顶指针top

// 入栈

```
int push(Stack *s, int value) {
    if (isFull(s)) {
        printf("栈满，无法入栈\n");
        return 0;
    }
    s->data[++(s->top)] = value;
    return 1;
}
```

- pop() 出栈函数

函数逻辑

- 判断栈是否为空
- 如果不为空，将栈顶元素弹出，并更新栈顶指针top

// 出栈

```
int pop(Stack *s, int *value) {
    if (isEmpty(s)) {
        printf("栈空，无法出栈\n");
        return 0;
    }
    *value = s->data[(s->top)--];
    return 1;
}
```

- peek() 获取栈顶元素函数

函数逻辑

- 判断栈是否为空
- 如果不为空，获取栈顶元素

// 获取栈顶元素

```
int peek(Stack *s, int *value) {
    if (isEmpty(s)) {
        printf("栈空，无法获取栈顶元素\n");
        return 0;
    }
    *value = s->data[s->top];
    return 1;
}
```

在主函数中的用法

```
int main() {  
    Stack s;  
    initStack(&s);  
  
    push(&s, 10);  
    push(&s, 20);  
    push(&s, 30);  
  
    int value;  
    if (pop(&s, &value)) {  
        printf("出栈元素: %d\n", value);  
    }  
  
    if (peek(&s, &value)) {  
        printf("栈顶元素: %d\n", value);  
    }  
  
    return 0;  
}
```

队列