链表

上机（是一个验证）

纸质版的答案

线性表： 数组， 链表（指针） 数据结构

基本概念

链表和数组都可用于存储数据，其中链表通过指针来连接元素，而数组则是把所有元素按次序依次存储。

不同的存储结构令他们有了不同的优势：

链表可以方便地删除、插入数据，操作次数是 O(1) 。但也因为这样寻找读取数据的效率不如数组高，在随机访问数据中的操作次数是 O(n) 。

数组可以方便的寻找读取数据，在随机访问中操作次数是 O(1) 。但删除、插入的操作次数却是却是 O(n) 次

顺序表

```
int a[10];  // a   a[1] = 2;   *(a + 1) = 2; * &
//
```

指针

什么是指针，我们一般都说 “它是一个地址”，“存储的是变量的地址”，“指向了一个值”

指针是什么：

1. 指针是一个普通的变量；
2. 既然指针是变量，那么肯定有自己的类型；
3. 既然指针是变量，那么肯定有自己的值；
4. 只不过指针的值跟一般变量的值不太一样，指针的值是一个“地址”。
5. 指针指向的数据

总结就是：指针就是一个普通的变量，有自己的类型和值，但是特殊的地方在于它的值和其它变量的值不一样，它的值是一个长的16进制的东西，即地址。

因为指针可以保存地址，计算机就可以利用指针保存任意大小的空间的首地址，可以随时改变指针里面的地址的值　　*可以取指针里面的内容　　&可以取变量里面的地址

构建链表

一个又一个的结点

```
typedef struct Node{
    int data;        // double float  但是它可以不是基本变量，它可以是个很复杂的东西
    struct Node* next;  // 这一行一般不会变
}Node;

Node* N1 = (Node*)malloc(sizeof(Node)); // 为什么我们这么创建
Node N2;
// -> .
// N1 是个 Node 的指针，可以指向任意的结点 可以换
```

构建链表实际上是构建了一个结点

一般我们描述的链表是什么？

1. 一个node型的指针　　// 头指针
2. 带空间的node型的指针　　// 带头节点的的头指针

关于链表的构建使用到指针的部分比较抽象，光靠文字描述和代码可能难以理解，建议配合作图来理解。

怎么构建链表：一个一个的将结点插入链表中，有两种插入的方法：头插法和尾插法，一般建议使用尾插法

1. 头节点，头指针
2. 头插法，尾插法

- 头插法　　// 插入的时候结点插入头节点和第一个结点，头指针指向它，它再指向第一个结点
- 尾插法呢　//node* tail;　　tail->next = temp; tail = temp;

单向链表

单向链表中包含数据域和指针域，其中数据域用于存放数据，指针域用来连接当前结点和下一节点

```
typedef struct Node {        //结点存储结构
  int value;         //数据域
  struct Node *next;         //指针域
}Node;
```

双向链表

双向链表中同样有数据域和指针域，不同之处在于指针域有左右（或上一个、下一个）之分，用来连接上一个节点、当前结点、下一个结点

```
typedef struct Node {
  int value;
  Node *left,*right;        // int a,* b, c;
}Node;
```

向链表中插入（写入）数据

单向链表

```
void insertNode(int i, Node *p) {
  Node* node = (Node *)malloc(sizeof(Node));        //new和
malloc
    // malloc
    // int b;   int* a; a = &b; *a = 1; <=> b = 1
    // int* a = (int*)malloc(sizeof(int)*10)
    // .  ->  // .    ->
    // Node node   node.value
    //
  node->value = i;
  node->next = p->next;     //头插法
  p->next = node;
}
```

具体过程如下：

上面介绍了简单的单向链表的插入数据，有时我们会将链表的头尾连接起来将链表变为循环链表：

```
void insertNode(int i, Node *p) {
  Node *node = (Node*)malloc(sizeof(Node));      //动态分配空间
  node->value = i;
  node->next = NULL;
  if (p == NULL) {       //  循环链表的特殊点
    p = node;
    node->next = node;
  } else {
    node->next = p->next;        //头插法
    p->next = node;
  }
}
```

由于是循环的链表，我们在插入数据时需要判断原链表是否为空，为空则自身循环，不为空则正常插入数据循环。具体过程可参考下面这张图。

双向循环链表的插入写法：0 = 0 = 0 1=

```
void insertNode(int i, Node *p) {
  Node *node = (Node*)malloc(sizeof(Node));
  node->value = i;
  node->next = NULL;         //. 和 ->
  if (p == NULL) {       //没有结点的情况
    p = node;
    node->left = node;
    node->right = node;
  } else {       //链表有结点的情况
    node->left = p;
    node->right = p->right;
    p->right->left = node;
    p->right = node;
  }
}
```

从链表中删除数据

单向（循环）链表

```
void deleteNode(Node *p) {
  p->value = p->next->value;
  Node *t = p->next;
  p->next = p->next->next;
  t->next = 0;
  free(t);
}
```

从链表中删除某个结点时，将 p 的下一个结点 (p->next) 的值覆盖给 p 即可，与此同时更新 p 的下下个结点。具体过程可参考下面这张图。

双向链表¶

```
void deleteNode(Node* p) {       //删除p结点
  p->left->right = p->right;
  p->right->left = p->left;
  p->left = 0; // 考试的时候 0 换成 null
  p->right = 0;
  free(p);
}
```

遍历链表 / / 改

```
void output(node* p){        //遍历链表(有头节点)
    if(!p)
        return;
    p = p->next;
    while(p){
        printf("%d\t",p->value);  // 可以换成很多操作
        p = p->next;
    }
}  // 函数的写法
```

翻转链表

p 1 -> 2-> 3 -> 4 -> 5

p -> 2 -> 3 -> 4 ->5　　temp -> 1

p -> 1 ->2 ->3 -> 4 ->5　temp ->2

p　2 -> 1 -> 3 -> 4 -> 5

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

void reverse(node* a) {
```

```
    // . ->       node a    .   a.next
    // node* a  a->next   int b   int* b
    // node* a;       int* b;   // 我们初始化两个值  a null  b 0
    // node* a = (node*)malloc(sizeof(node));  123456   123455
    // a->data = 1;
    // node a;        int* b;
    // a.data = 1;
    // 1. . ->
    // 2. 指针能改变地址
    if(!a || a->next == null){
        return;
    }
    node* p = a->next;
    while(p) {
        node* temp = p->next;
        node* first = a->next;
        if(p != first) {    // 防止 1 指向自己了
            p->next = first;
            a->next = p;
        }
        p = temp;
    }
}
```

```
void reverse(node* p){       //翻转链表，头插法  没有消耗其他的空间
    if(!p||!p->next)         //链表不存在或者没有结点的情况
        return;
    node* first = p->next,*temp = p->next->next;
    while(temp){        //遍历链表
        first->next = temp->next;        //将temp提出来
        temp->next = p->next;       //头插法
        p->next = temp;
        temp = first->next;
    }
}


```

二重指针

任务

- 尝试写一下循环链表的构建、插入、删除和搜索的代码

```
typedef struct LNode{        //结点存储结构
    int data;
    struct LNode* next;
}LNode;
bool insertLNode(LNode*p,int i){     //头插法插入i进链表
    if(!p)
        return false;
    LNode* temp = (LNode*)malloc(sizeof(LNode));
    temp->data = i;
    temp->next = p->next;        //头插法
    p->next = temp;
    return true;
}
bool deteleLNode(LNode* p){      //删除p后一结点
    if(!p||p->next==p)       //p不存在或者没有结点
        return false;
    LNode* temp = p->next;
    p->next = temp->next;
    temp->next = NULL;
    free(temp);      //释放temp结点
    return true;
}
bool search(LNode* p,int i){         //搜索链表p中是否存在值为i的结点
    if(!p||p->next==p)       //p不存在或者没有结点
        return false;
    LNode* temp = p->next;       //temp现在指向第一个结点
```

```
    while(temp->next!=p){        //遍历链表
        if(temp->data==i)
            return true;
        temp = temp->next;
    }
    return true;
}
```

- 尝试写一下双向链表的构建、插入、删除和搜索的代码

```
typedef struct DNode{        //双链表结构
    int data;
    struct DNode* prior,* next;
}DNode;

bool insertDNode(DNode* p,DNode* s){    //将s插入到p之后
    if(!p)
        return false;
    s->next = p->next;
    p->next->prior = s;
    s->prior = p;
    p->next = s;
    return true;
}
bool deleteDNode(DNode* p){     //删除p的后面个结点
    if(!p)
        return false;
    DNode* q = p->next;
    if(!q)
        return false;
    p->next = q->next;       //删除结点操作
    if(q->next != NULL)
        q->next->prior = p;
    q->next = NULL;
    free(q);         //将q的空间释放
    return true;
}
DNode* search(DNode* p,int target){     //搜索等于target的结点
    if(!p)
        return 0;
    p = p->next;
    while(p){        //遍历链表寻找目标结点 0 false null
        if(p->data == target)
            return p;
        p = p->next;
    }
    return 0;
```

```
}

```

- 做下面的题：

1.

若线性表用单链表(带表头结点)作为存储结构，写出其就地逆置算法。(本题 15 分)

```
// 算法思想： 遍历单链表，从第一个开始，每一个结点重新插入到了新的链表中的第一个位置，即头插法，最终就可以实现单链表的就地逆置

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

void reverse(Node* p) { // 没有重新改变头节点的地址
    if(!p)  // 指针不存在
        return;
    Node* first = p->next;
    Node* q = first->next;
    while(q) {      // 遍历结点，逐个插入
        Node* temp = second->next;  // 保存第三个结点的地址
        q->next = first;        // 头插法
        first->next = temp;         // 接上第三个结点
        first = q;          // first 改变了
        q = temp;
    }
}
```

```
Node* reverse2(Node* p) {
    if(!p){     // 判断链表是否存在
        return 0;
    }
    Node* res = (Node*)malloc(sizeof(Node));
    Node* temp = p->next;
    while(temp) {   // 遍历链表，逐个头插法
        Node* temp2  = temp->next;

        temp->next = res->next;     // 头插法
        res->net = temp;

        temp = temp2;
    }
```

```
    return res;
}
```

```
// 算法思想： 先判断链表，如果链表结点为0或者1 直接返回，否则从第二个结点开始逐个将其用头插法的方法重新插入一遍
typedef struct Node{
    int value;
    struct Node* next;
}Node;

void reverse(Node* temp) {  // 链表带头节点， 将其逆置
    if(!temp || temp->next) {       // ,,,
        return;
    }
    Node* first = temp->next, * second = temp->next->next;
    while(second) {
        Node* p = second;
        p->next = first;    // 头插法
        temp-> = p;
        first = p;
        second = second->next;
    }
}
```

```
typedef struct node{        //存储结构
    int data;
    struct node* next;
}node:
```

```
5
6  void reverse(node* linklist){
7      if(!linklist||p->next)       //链表不存在的情况
8          return;
9      node* first = linklist->next,*temp = linklist->next->next;
10     while(temp){
11         first->next = temp->text;        //取出temp
12         temp->next = linklist->next;     //头插法
13         linklixt->next = temp;
14         temp = first->next;      //下一个结点
15     }
16 }
17
```

2.

x y z z y x

1 2 1 2 2 1 2 1

栈、队列　先进后出　x y z z y x x y z z y x

设单链表中存放 n 个字符，试设计一个算法，使用栈判断该字符串是否中心对称，如 xyzzyx 即为中心对称字符串。（小题 15 分）

```
1  typedef struct node{         //链表结点的存储结构
2      char data;
3      struct node* next;
4  }node;
5
6  bool function(node* linklist){       //带头结点的链表
7      if(!linklist)
8          return false;
9      char stack[100];         //假设字符串不超过100个字符,构建一个栈
10     int top = -1;        // 栈的写法
11     node* p = linklist->next;        //p指向第一个结点
12     while(p){        //遍历链表
13         if(top==-1||stack[top]!=p->data)     //入栈的情况
14             stack[++top] = p->data;
15         else if(stack[top]==p->data)         //出栈的情况
16             top--;
17         p = p->next;
18     }
19     if(top==-1)      //判断栈是否为空
20         return true;
21     else
22         return false;
23 }
```

3.

编写一算法，以完成在带头节点单链表 M 中第 n 个位置前插入元素 X 的操作。（ 20 分）

```
typedef struct node{          //存储结点
    int data;
    struct node* next;
}node;

void insert(node* M,int X,int n){        //链表的结点数大于n
    if(!M||n<=0)          //链表不存在或者n不合理
        return;
    node* temp = (node*)malloc(sizeof(node));        //初始化结点
    temp->data = X;
    temp->next = NULL;
    node* p = M->next;
    for(int i=0;i<n-1;++i)       //找到n-1的位置
        p = p->next;
    temp->next = p->next;        //插入temp
    p->next = temp;
}
```

4.

已知非空线性链表第一个节点由 list 指出，请写一个算法交换 p 所指的节点与其下一个节点在链表中的位置(设 P 指向的不是链表最后那个结点)。(本题 20 分)。

```
typedef struct ndoe{          //节点存储结构
    int data;
    struct node* next;
}node;
bool swap(node* p,node* list){       //交换p和下一个节点的顺序
    if(!p||!list)
        return false;
    if(p==list){          //p为第一个节点的情况
        list = p->next;
        p->next = list->next;
        list->next = p;
        return true;
    }
    node* temp = list;
    while(temp->next!=p)          //找到p的前一个节点
        temp = temp->next;
    temp->next = p->next;         //交换p和后一节点
    p->next = temp->next->next;
    temp->next->next = p;
```

```
    return true;
}
```

5.

请用链表编程实现：从键盘读入整数，并按从小到大的顺序输出输入整数中互不相等的那些整数。（本题 20 分）。

算法思想：每次读入整数i，将整数 i 找到在链表中的某一个位置，该位置满足前面的数都比 i 小，后面的数都比 i 大，如果存在和 i 相等的数就不插入，否则就在该位置插入值为 i 的结点，每一个整数都这么插进链表中，最后就能得到从小到达互不相等的整数的链表，然后输出即可。

```
typedef struct node{         //节点存储结构
    int data;
    struct node* next;
}node;
bool insert(node* p,int i){ //在链表p中按从小到大的顺序插入不相等的整数
    if(!p)
        return false;
    node* temp1=p,*temp2=p->next;
    while(temp2&&i>temp2->data){         //找到一个位置，temp1前面的数都比i小，temp2的数大于等于i
        temp1 = temp1->next;
        temp2 = temp2->next;
    }
    if(temp2&&i==temp2->data)        //相等的就不插入了
        return false;
    node* temp = (node*)malloc(sizeof(node));    //初始化i
    temp->data = i;
    temp->next = NULL;
    if(!temp2)  //应该插入最后个节点
        temp1->next = temp;
    else{        //中间插入的情况
        temp->next = temp2;
        temp1->next = temp;
    }
    return true;
}
void output(node* p){        //输出链表
    if(!p)
        return;
    p = p->next;
    while(p){
        printf("%d\t",p->data);
        p = p->next;
    }
```

```
    printf("\n");
}
int main(){
    node* p = (node*)malloc(sizeof(node));       //初始化一个链表
    p->next = NULL;
    int i;
    while(i){        //从键盘中读数，为0的情况下退出
        printf("Enter i:\n");
        scanf("%d",&i);
        insert(p,i);
    }
    printf("list:\n");
    output(p);
}
```

6.

定义一个双向循环链表，并写出其定位、插入和删除算法。

```
//同第一个代码一样
```

7.

假设有两个按元素值递增有序排列的线性表 A 和 B，均以单链表作存储结构，请编写算法将表 A 和表 B 归并成一个按元素值非递减有序（允许值相同）排列的线性表 C，并要求利用原表（即表 A 和表 B）的结点空间存放表。（本题 20 分）

```
typedef struct node{
    int data;
    struct node* next;
}node;
node* merge(node* A,node* B){        //有序融合A,B
    node* c=A,*pa=A->next,*pb=B->next; //A作为C表表头
    while(pa&&pb){       //A,B都还有节点的情况
        if(pa->data<=pb->data){      //插入pa
            c->next = pa;
            c = pa;
            pa = pa->next;
        }
        else{        //插入pb
            c->next = pb;
            c = pb;
            pb = pb->next;
        }
    }
    if(pa)       //将A或者B剩余的节点插入
        c->next = pa;
```

```
    else if(pb)
        c->next = pb;
    return A;       //因为将A作为了C表的表头
}
```

8.

已知线性表中的元素以值递增有序排列，并以单链表作存储结构。编写程序，删除表中所有值大于 mink 且小于 maxk 的元素，同时释放被删除的结点空间（本题 15 分）

```
typedef struct node{
    int data;
    struct node* next;
}node;
bool delete(node* l,int mink,int maxk){     //```
    if(!l)
        return false;
    if(l->data<=maxk)       //第一个节点就不满足的情况
        return true;
    node* temp;
    while(l&&l->data<=mink)     //找到第一个符合条件的节点
        l = l->next;
    while(l&&l->data<maxk){
        temp = l;       //提出temp
        l = l->next;
        temp->next = NULL;
        free(temp);     //释放temp空间
    }
    return true;
}
```

9.

a 和 b 是两双向链表。其中每一个结点存放一个整数。试编函数，将链表 b 和链表 a 合并，且去除其中整数值相同的结点，返回合并后的链表首地址。（本题 15 分）

```
typedef struct node{        //双链表结构
    int data;
    struct DNode* prior,* next;
}node;
void insert(node* c,node* temp){    //将temp有序插入c
    if(!c||!temp)
        return;
    node* p = c->next;
```

```
    while(p&&p->data<temp->data){        //找到插入的位置
        p = p->next;
        c = c->next;
    }
    if(p->data==temp->data)      //排除相等的整数结点
        return;
    if(! p)       //尾插法
        c->next = temp;
    else{
        temp->next = c->next;
        c->next = temp;
    }
}
node* merge(node* A,node* B){        //有序融合A,B，并去除其中整数值相同的结点
    node* temp,*pa=A->next,*pb=B->next;
    node* c = (node*)malloc(sizeof(node));
    c->next = NULL;
    while(pa){       //一个一个将a表的结点插入c
        temp = pa;
        pa = pa->next;
        temp->next = 0;
        insert(c,temp);
    }
    while(pb){
        temp = pb;
        pb = pb->next;
        temp->next = 0;
        insert(c,temp);
    }
    return c;
}
```